FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Statistical Comparison of Different Machine-Learning Approaches for Malaria Parasites Detection in Microscopic Images

Mafalda Falcão Torres Veiga de Ferreira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Filipe Teixeira, Assistant Professor

Second Supervisor: Luís Filipe Caeiro Margalho Guerra Rosado, MSc

July 25, 2017

Contact Information:

Mafalda Falcão Ferreira
ei12036@fe.up.pt

Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias
4200-465 Porto
Portugal

"Statistical Comparison of Different Machine-Learning Approaches for Malaria Parasites Detection in Microscopic Images"

Copyright © Mafalda Falcão Ferreira, 2017.
All rights are reserved.

Statistical Comparison of Different Machine-Learning Approaches for Malaria Parasites Detection in Microscopic Images

Mafalda Falcão Torres Veiga de Ferreira

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2017

Abstract

Malaria is a severe public health problem across the world, particularly in developing countries (≈80% of the cases occur in Africa), putting at special risk the most unprotected groups of society: children and pregnant women.

Since it can be caused by 4 different species of parasites, each having different stages of evolution, approaching the right diagnosis without access to costly equipment is complex. Ergo, research has focused on speeding up and lowering the costs of its diagnosis, by resorting to automatic machine classification of microscopic images.

Still, most approaches rely on simplistic, single-model classifiers, with a constant absence of a systematic statistical comparison in the literature that supports a particular technique or feature.

Hence, this dissertation aims to: (i) design and execute a statistical comparison of different ML approaches for the detection of such parasites in microscopic images, (ii) identify which features are more relevant for prediction, and (iii) identify which models and techniques achieve the best results, balancing precision and recall.

Given the stated problem, and before approaching it, we performed a statistical analysis on the dataset, to discover its proportions and to detect highly correlated features.

After knowing the data, it was developed a framework that (i) optimizes the values of the considered classification and feature selection algorithms, (ii) computes a statistical comparison of different machine learning approaches to the same dataset, using different metrics on the *cross validation*, in which were studied different metrics to measure the performance value variation and evaluate which one is consistent with the data, and, finally, (iii) performs a statistical hypothesis test, to guarantee that the data model with the best performance is distinct from all the others considered in this study. As result, one can verify an improvement over the established baseline, by using a ***False Discorery Rate*** **feature selection method followed by a *Ada Boosting* classifier with 350 estimators**.

Acknowledgements

I would first like to thank my supervisor Prof. Luís Filipe Teixeira for all the support, guidance and orientation, during my research and realization of this dissertation.

I would also like to thank to my co-supervisor Luís Rosado and to the Fraunhofer AICOS Association for the given help and amazing conditions to elaborate this dissertation, during this semester.

I would also like to Prof. Chris Williams and Doctor John A. Quinn from the University of Edinburgh, for their involvement in the research phase, giving me insights on the biology of the disease and the technologies already applied to this domain.

Finally, I must express my gratitude to my family and specially to Hugo, for providing me with unfailing support and continuous encouragement and motivation throughout this year, to finish up this dissertation as well as I could. This accomplishment would not have been possible without them. Thank you.

Mafalda Falcão Torres Veiga de Ferreira

"I believe that at the end of the century the use of words
and general educated opinion will have altered so much that one
will be able to speak of machines thinking without expecting to be contradicted."

Alan Turing

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

ANN	ARTIFICIAL NEURAL NETWORK(s)
APR	Average Precision
BAG-DT	BAGGED DECISION TREE(s)
BEP	Break Even Point
BST-DT	BOOSTED DECISION TREE(s)
BST-STMP	BOOSTED STUMPS
CNN	Convolutional Neural Network(s)
DL	Deep Learning
DT	Decision Tree(s)
FN	False Negatives
FP	False Positives
FSA	Feature Selection Algorithms
FSC	F-Score
GP	Genetic Programming
HSA	Heuristic Search Algorithms
kNN	K-NEAREST NEIGHBORS
MCE	Minimum Cross Entropy
ML	Machine Learning
MLP	Multi-Layer Perceptron
MP	Malaria Parasites
NB	NAÏVE BAYES
PCA	Principal Component Analysis
RBC	Red Blood Cells
RF	RANDOM FOREST(s)
ROC	Receive Operating Characteristics
RMSE	Root Mean Squared Error
SL	Supervised Learning
SSA	Sequential Selection Algorithms
SVM	SUPPORT VECTOR MACHINE
TN	True Negatives
TP	True Positives
USL	Unsupervised Learning

Chapter 1

Introduction

This chapter serves to present the problem and the scope of this dissertation. Section 1.1 describes the context of the problem. Section 1.2 (p. 2) defines the main issues not yet solved. This section also explains the reasons that motivate this work and the goals that have been set as an end result. Section 1.4 (p. 3) details the full report structure, specifying the topics of each chapter.

1.1 Context

Since 2000, substantial progress has been made in fighting malaria. According to the latest estimates, between 2000 and 2015, malaria case incidence was reduced by 41% and malaria mortality rates by 62%. At the beginning of 2016, malaria was considered to be endemic in 91 countries and territories, down from 108 in 2000. Much of the change can be attributed to the wide-scale deployment of malaria control interventions. Despite this remarkable progress, malaria continues to have a devastating impact on people's health and livelihoods. Updated estimates indicate that 212 million cases occurred globally in 2015, leading to 429 000 deaths, most of which were in children aged under 5 years in Africa. In the same year, nearly half of the world's population was at risk of malaria. Most malaria cases and deaths occur in sub-Saharan Africa. However, South-East Asia, Latin America and the Middle East, are also at risk.

Some population groups are at considerably higher risk of contracting malaria, and developing severe disease, than others. These groups include infants, children under 5 years of age, pregnant women and patients with HIV/AIDS, as well as non-immune migrants, mobile populations and travelers. [Org16]

Table 1.1: Malaria Parasites that infect Humans.

Species	**Intra-RBC Schizont Period**	**Type of RBC**	**Global Distribution**
P. vivax	48 hours	Retivulocytes	Everywhere except Africa
P. ovale	48 hours	Retivulocytes	Africa
P. malarie	72 hours	Older RBC	Everywhere
P. falciparum	48 hours ($\approx$)	All	Tropical Regions

Five species of protozoan parasites of the genus *Plasmodium* infect humans: *P. falciparum*, *P. vivax*, *P. ovale* and *P. malariae*. Although *P. vivax* is the most widespread form of malaria infection in the world, *P. falciparum* causes the most severe disease and is responsible, by far, for most deaths and serious morbidity due to malaria.

The evolution of a parasite is really complex. The species undergoes multiple transformations within the mosquito and human host; at least a dozen different steps have been defined [Ken12]. Nevertheless, it is important to correctly identify with which species a person is infected with, because different parasites have different life cycles.

Given the complexity of the disease, it is crucial to reach the right diagnosis to save the lives of millions of people. However, getting to the right diagnosis means having the right and the expensive equipment whenever it is needed. In most of the cases, as the disease has more impact in Africa and in the developing countries, it may not be possible (in fact, it is quite impossible) for these poor areas to afford this kind of medical materials.

1.2 Motivation and Goals

In order to tackle the scenario described in Section 1.1 (p. 1), research has been focused on improving the diagnosis procedures, by speeding up the all process and reducing the costs of the equipments used and needed. Here is where this work fits in: with the help of technology, this better and cheaper solutions will, hopefully, reach everywhere, specially the most needed countries. For this project, the following high-level goals have been defined:

- Design and execute a comparison, on a statistic level, of different machine learning approaches for the detection of the parasites in microscopical images;
- Identify which is/are the more relevant feature(s) to the prediction;
- Identify which machine learning model/technique achieves the best results.

A more formal presentation of these goals, and the way they connect to the fundamental research questions, can be found in Chapter 4 (p. 41).

1.3 Publications

The work and ideas covered in this dissertation have been target of one scientific publication, currently submitted and under review:

1. Mafalda Falcão Ferreira, Luís F. Teixeira, Luís Rosado, "Improving Malaria Parasites Detection in Thick Blood Smear Images: A Statistical Approach" at the *9th Symposium in Informatics* (INForum), 2017.

1.4 Document Structure

The remainder of this document is structured as follows:

- Chapter 2, "Technical State of the Art" (p. 5) provides a technical background, particularly in Machine Learning, without going into specifics about the detection of malaria parasites;
- Chapter 3, "State of the Art in Image Detection of Malaria Parasites" (p. 29) provides a literature review concerning the specific domain of this project;
- Chapter 4, "Thesis Proposal" (p. 41) describes the dissertation proposal given the problems identified during the literature review, and choses a baseline work for our development; finally, we draft the main research questions along with the primary goals.
- Chapter 5, "Statistical Comparison of Different Approaches" (p. 47) describes our primary study of identifying and statistically comparing different feature selectors and classifiers to improve the detection of malaria parasites in microscopical images;
- Chapter 6, "Implementation Details" (p. 63) presents a series of technological details regarding the technological stack and the framework's architecture;
- Chapter 7, "Conclusions" (p. 69) concludes this dissertation with an overview of the research questions asked and the main contributions, by essentially doing an overall reflexion over the observed results, while proposing guidelines for possible future work.

Chapter 2

Technical State of the Art

This chapter describes the technical State of the Art. Section 2.2 (p. 6), Section 2.3 (p. 15) and Section 2.4 (p. 16) introduce existing Machine Learning algorithms, approaches and techniques, defining some important formalizations that are the foundation for this dissertation. Section 2.5 (p. 21) summarizes statistical comparison works applied to different domains. Section 2.6 (p. 23) presents some techniques commonly used for image processing and feature extraction. Finally, Section 2.7 (p. 23) overviews some of the most interesting and used applications and frameworks that help the task of Data Mining and Machine Learning.

2.1 What is Machine Learning?

Machine Learning (ML) is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in Artificial Intelligence (AI). It is a method of data analysis that automates analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look. [Sys16; Mur12]

The models are constantly exposed to new data, which makes the iterative factor an important aspect of ML approaches. They learn from previous computations to produce reliable, repeatable decisions and results [Mur12]. ML is not new, but it has gained a fresh momentum, because of the computing technologies evolution [Sys16]. Nowadays, the ability to automatically apply complex

mathematical calculations to Big Data[1] — in a repetitive and fastest way — is a recent achievement on the area.

The increasing interest on ML is due to the same factors: (1) growing volumes and varieties of available data, (2) computational processing that is cheaper and more powerful, and (3) affordable data storage. They all point to the capacity of producing models that can handle — *i.e.* analyze and deliver faster and more accurate results — big volumes of data, quicker and automatically. This models will lead us to high-value predictions that can guide better decisions and smart actions in real time, minimizing human intervention during the training process. [Sys16]

It is important to underline the difference between *Machine Learning* and *Data Mining* (DM), because its definitions are often mixed up, even by experts in each field:

- **Data Mining** discovers previously unknown patterns and knowledge;
- **Machine Learning** is used to reproduce known patterns and knowledge, automatically apply that to other data, and then automatically apply those results to decision making and actions.

The increased power of today's computers has also helped Data Mining techniques evolve for usage in ML. They are, however, two distinct areas. Machine Learning is further divided into two[2] main areas: **Supervised Learning** (*cf.* § 2.2), and **Unsupervised Learning** (*cf.* § 2.3, p. 15).

2.2 Supervised Learning

Supervised Learning is the task of inferring a function from labeled training data. The training data consists in a set of training examples. In supervised learning, an example is a pair compose by an input object and an expected output value. A supervised learning algorithm analyzes the trained data and produces an inferred function, that it is used to map new examples. The best scenario would be for the algorithm to correctly determine the class labels for unseen instances and, for this, it is required a learning algorithm to generalize from the training data to all the situations, in a "reasonable" way. [Wikn; Mur12]

Currently, many of our day-to-day applications are powered by ML algorithms, including:

- Fraud Detection
- Web Search Results
- Real-time Ads on Web Pages
- Text-based Sentiment Analysis
- Credit Scoring and Next-Best Offers
- Prediction of Equipment Failures
- New Pricing Models
- Network Intrusion Detection
- Pattern and Image Recognition
- Email Spam Filtering

[1] Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.

[2] There are other ML areas, such as *Reinforcement Learning*, which are out of scope of this dissertation.

2.2.1 Binary Classification

In ML, *Classification* is the problem of identifying to which of a set of categories a new observation belongs to, based on the training set of the data containing observations, whose category membership is already known. Classification is an example of pattern recognition.

In the ML domain, classification is considered a field of SL. In others words, it is about predicting class labels given input data. In binary classification, there are two possible output classes. In multi-class classification, there are more than two possible classes [Wikm; Mur12], as it will be discussed in Section 2.2.2.

Classification algorithms are known as *classifiers*. Section 2.2.3 (p. 9) describes some of these algorithms.

2.2.2 Multi-Class Classification

As referred by Tsoumakas and Katakis [TK07], traditional single-label classification is concerned with learning from a set of examples that are associated with a single label l from a set of disjoint labels L, where $|L| > 1$. If $|L| = 2$, then the learning problem is considered a binary classification problem; if $|L| > 2$, then it is a multi-class classification problem.

In the past, multi-label classification was mainly motivated by text categorization and medical diagnosis. Nowadays, we notice that multi-class classification methods are increasingly required by modern applications, such as protein function classification, music categorization and even semantic scene classification. [TK07]

We can group the already existing multi-class classification methods into two main categories:

1. **Problem Transformation Methods**, which transform multi-label classification problem either into one or more binary classification or into a regression problem;

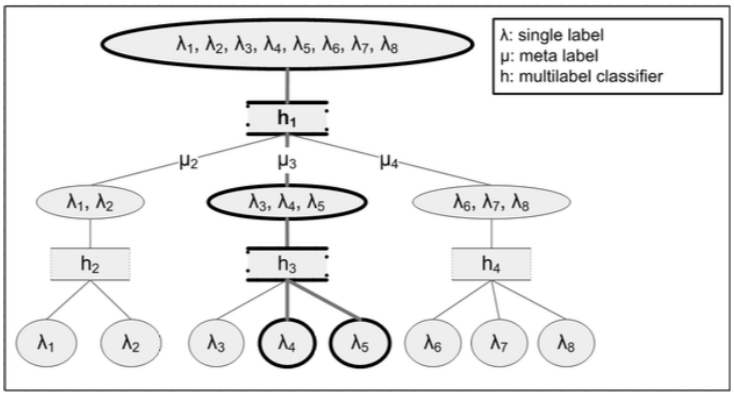
2. **Algorithm Adaptation Methods**, which extend specific learning algorithms to handle the multi-class data directly.

Tsoumakas and Katakis studied two different algorithms that fit on the different categories defined above, testing their effectiveness and efficiency on domain with a large number of labels: *the HOMER Algorithm* and *the Balance k-Means* [TKV08].

The HOMER (Hierarchy Of Multilabel classifiERs) algorithm follows the paradigm of the algorithm design *divide-and-conquer* and its main goal is to transform a multi-class classification task, with a large set of labels L, into a tree-shaped hierarchy of simpler multi-label classification tasks, each one dealing with a small number $k < |L|$ of labels.

Each node n of this tree contains a set of labels $L_n \subseteq L$. There are $|L|$ leaves, each one containing a singleton (single element set) λ_j with a different label λ_j of L. Each internal node n contains the union of the label sets of its children, $L_n = L_c$ for $c \in children(n)$. The root contains all labels, $L_{root} = L$. Figure 2.1a (p. 8) exemplifies the above terminology.

At the end of this process, HOMER constructs a hierarchy of multilabel classifiers, each dealing with a smaller set of labels compared to L, and a more balanced example distribution. The



(a)

Input: number of clusters k, labels L_n, label data W_i, iterations it
Output: k balanced clusters of labels
for $i \leftarrow 1$ **to** k **do**
  // initialize clusters and cluster centers
  $C_i \leftarrow \emptyset$;
  $c_i \leftarrow$ random member of L_n ;
while $it > 0$ **do**
  foreach $\lambda \in L_n$ **do**
    for $i \leftarrow 1$ **to** k **do**
      $d_{\lambda i} \leftarrow \text{distance}(\lambda, c_i, W_i)$
    finished $\leftarrow$ false;
    $\nu \leftarrow \lambda$;
    while *not finished* **do**
      $j \leftarrow \arg\min_i d_{\nu i}$;
      Insert sort (ν, d_ν) to sorted list C_j;
      if $|C_j| > \lceil |L_n|/k \rceil$ **then**
        $\nu \leftarrow$ remove last element of C_j;
        $d_{\nu j} \leftarrow \infty$;
      else
        finished $\leftarrow$ true;
  recalculate centers;
  $it \leftarrow it - 1$
return $C_1, ..., C_k$;

(b)

Figure 2.1: (a) HOMER - Sample hierarchy for a multilabel classification task with 8 labels; (b) BALANCE K MEANS, in pseudocode.

authors claim improved predictive performance along with linear training and logarithmic testing complexities with respect to $|L|$.

The BALANCE K MEANS is a balanced clustering algorithm, which extends the well-known K MEANS algorithm with an explicit constraint on the size of each cluster. Figure 2.1b describes how the algorithm works in pseudocode.

The key element in the BALANCE K MEANS algorithm is that, for each cluster i, the list of labels C_i is maintained, sorted in ascending order of distance to the cluster centroid c_i. When the insertion of a label into the appropriate position of the sorted list of a cluster causes its size to exceed the

maximum allowed number of labels (approximately equal to the number of items divided by the number of clusters), the last (furthest) element in the list of this cluster is inserted to the list of the next most proximate cluster. This may lead to a cascade of $k-1$ additional insertions, in the worst case scenario. Another difference, compared to K MEANS, is that the researchers limited the number of iterations using a user-specified parameter, *it*, as no investigation of convergence was attempted.

2.2.3 Machine Learning Classifiers

Classifiers are classification algorithms, as referred in Section 2.2.1 (p. 7), and are an instance of Supervised Learning. A classifier can also have another role: it can often point to a mathematical function, implemented by the classification algorithm, that matches the input value to a certain category.

There are several different classifications algorithms, such as:

- Decision Trees
- Neural Networks
- Genetic Algorithms
- Naive Bayes
- SVM
- (...)

This section will describe some widely used algorithms, some of them previously found in the literature for Malaria Parasites detection.

2.2.3.1 Decision Trees

Decision Trees are methods that construct a model of decisions based on actual values of attributes in the data. These trees are usually fast and accurate, representing a popular technique in ML. [Kot07; Mur12]

Random Forests

Random forests — or Random Decision Trees — are an ensemble learning method of classification, regression and other tasks. This kind of algorithms construct a great number of decision trees at the training time, outputting the label that is the mode most frequent value in a data set of that particular class. Random decision forests correct the decision trees habit of overfitting their training set. [Wikl; Mur12]

As explained by Breiman and Cutler [BC], it was shown that the forest error rate depends on two factors:

- The **correlation between any two trees in the forest**, where increasing correlation increases the forest error rate;
- The **strength of each individual tree in the forest**, where a tree with a low error rate is a *strong classifier*.

When the training set for the current tree is drawn by sampling with replacement, about one third of the cases are left out of the sample. This *out-of-bag* data is used to get a running impartial estimate of the classification error, as trees are added to the forest.

After each tree is built, all of the data is run down the tree and proximities are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by each one being divided by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data. [BC]

Extremely Randomized Trees

EXTRA-TREES consists on building an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other tree-based ensemble methods are: 1) it splits nodes by choosing cut-points fully at random and 2) it uses the whole learning sample to grow the trees. As we can see on Figure 2.2, it has two parameters: K, the number of attributes randomly selected at each node and *nmin*, the minimum sample size for splitting a node. It is used several times with the original learning sample to generate an ensemble model (M is the number of trees of this ensemble). The predictions of the trees are aggregated to yield the final prediction, by majority vote in classification problems. [GEW06]

Split_a_node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
– If **Stop_split**(S) is TRUE then return nothing.
– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;
– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = $ **Pick_a_random_split**(S, a_i), $\forall i = 1, \ldots, K$;
– Return a split s_* such that Score(s_*, S) $= \max_{i=1,\ldots,K}$ Score(s_i, S).

Pick_a_random_split(S,a)
Inputs: a subset S and an attribute a
Output: a split
– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;
– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
– Return the split $[a < a_c]$.

Stop_split(S)
Input: a subset S
Output: a boolean
– If $|S| < n_{\min}$, then return TRUE;
– If all attributes are constant in S, then return TRUE;
– If the output is constant in S, then return TRUE;
– Otherwise, return FALSE.

Figure 2.2: EXTRA-TREES Algorithm, in Pseudocode (*for numerical attributes*).

2.2.3.2 Naïve Bayes

The NB CLASSIFIER provides a simple approach to represent, use and learn probabilistic knowledge. The method's performance goal is to accurately predict the class of test instances and in which the training instances include information [JL95; Kot07]. It is *naive* in the sense that it relies in two important *simplifying* assumptions:

1. It assumes that the predictive attributes are **conditionally independent**, given a certain class.
2. It posits that **no hidden or latent attributes influence** the prediction procedure.

As one can learn in John and Langley [JL95], these assumptions support many qualified algorithms for classification and learning. For example, given a classification problem or a certain instance C, let X be a vector $X = X_1, \ldots, X_n$ that represents some n features and k the possible outcomes or classes (C_k). Using the *Bayes' theorem*, the conditional probability can be decomposed as:

$$p(C_k|X) = \frac{p(C_k) \times p(X|C_k)}{p(X)}$$

Or, colloquially speaking and using the right terminology, it can be translated to:

$$posterior = \frac{prior \times likelihood}{evidence}$$

2.2.3.3 Neural Networks

An *Artificial Neural Network* is an information processing paradigm. It is inspired in the way the biological nervous systems work. The key element of ANN is the structure of the information processing system. The network can be described as a big number of highly and internally connected elements - named *neurons* - working together to solve a specific problem. Just like us, humans, ANN learn by example and, identical to the process of learning in biological systems, learning in a ANN involves adjustments to the synaptic connections already existent between the neurons. [SS]

Neural Networks can derive meaning from imprecise data. This ability can be used, for example, to detect some tendencies that are imperceptible to either people or other computational techniques. Thus, a trained NN can be seen as an *expert* in a certain category of the given information for analysis. Ergo, as defined by Stergiou and Siganos [SS], pointing the following advantages:

1. It can be used to **provide projections**, given new and unknown situations, answering to *what if* questions;
2. It has an **adaptive learning** ability - i.e. ability to learn how to perform task, given a training/initial experience;
3. It has a **self-organization** ability - i.e. this kind of networks can create their own organization/representation, given a certain information received;
4. It operates in **real time**;
5. It is **fault tolerant**, via redundant information coding.

It should be noted that many of these advantages are not unique to Neural Networks, but shared amongst many machine learning approaches.

Perceptron Networks

A perceptron network, with a **single layer**, can shortly be describe as follows: If X_1 through X_n are input feature values and $W_1, \ldots, W_n$ are connection weights/prediction vector (typically real numbers, between -1 and 1), then perceptron computes the sum of weighted inputs:

$$\sum_i Xi,\ Wi \tag{2.1}$$

And output goes through an adjustable threshold. If the $\sum$ is above the threshold, the output is equal to 1; else it is equal to 0. The perceptron algorithm is commonly used to learn from an amount of training instances, being run, repeatedly, through the all set until it finds a prediction vector that fits the training set. This rule is then used to predict the labels on the test set.

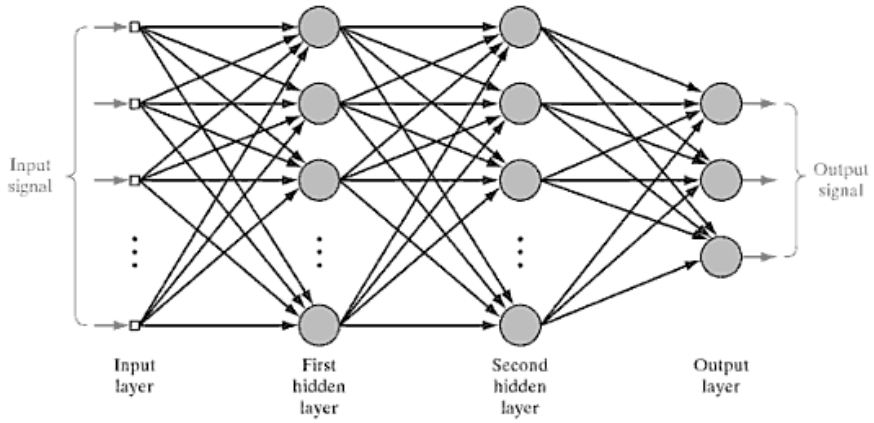


Figure 2.3: Architectural Graph of a Multilayer Perceptron Network, with Two Hidden Layers.

Perceptrons can only classify linearly separable sets of instances. If the instances are not linearly separable, the perceptron learning capacity will never reach a point where all instances are classified properly. Multilayered perceptrons were created in an attempt to solve this problem. A multi-layer neural network consists in a enormous number of neurons joined together in a pattern of connections. The neurons, in the network, can be classified by: *input units* (neurons that received the data/information), *output units* (where it is possible to find the results of the information processing) and *hidden units* (neurons in between). Feed-Forward ANN allows signals to travel the network one way only (beginning the *journey* at the input and ending it at the output). [Kot07].

Deep Learning

A Deep Learning (DL) network is just like a multi-layer neural network, but it has some distinct characteristics. Its architecture has more hidden layers than the usual NN. Also, the connectivity between neurons remains the same in a *normal* NN, differing from a DL network, where the connectivity can be different in every layer. The types of neurons connectivity are:

- **Dense.** There is fully connection between neurons of the different layers.
- **Convolutional.** Instead of a full connection between layers, convolutional networks try to capture "locality" and "proximity" by only connecting neurons to "nearby" features. [Unia]
- **Max Pool.** In the case of image processing, after obtaining the convolved features, the size of the region is decided to pool the convolved features over. Then, the features are divided

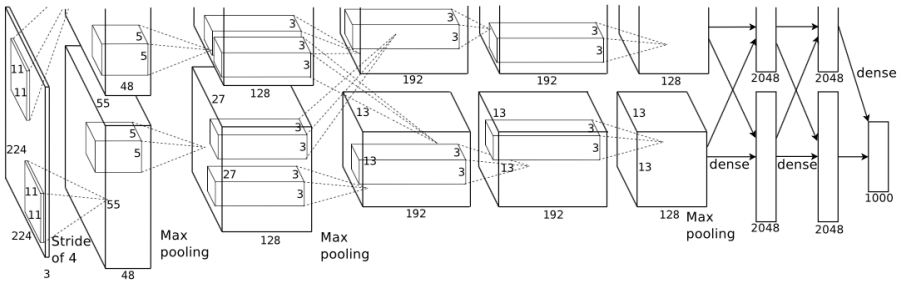


Figure 2.4: An illustration of the architecture of a CNN, explicitly showing the delineation of responsibilities between the two *Graphics Processing Units* (GPU). One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. *Source*: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

into disjoint regions, according to the established size, and take the mean (or maximum) feature activation over these regions to obtain the pooled convolved features. These pooled features can then be used for classification. [Unib]

2.2.4 Meta Learning

Meta Learning is another subfield of ML. This automatic learning algorithms try to replace humans in the process of creating diverse computational learning models from data. Once fed with new data and the goals description, meta learning systems should be able to "make decisions" in classification, regression, association tasks and/or give comprehensible models of data.

Meta Learning algorithms that "learn how to learn" and guide model selection have been advanced in the fields of statistics, ML, Computational Intelligence, and Artificial Intelligence.

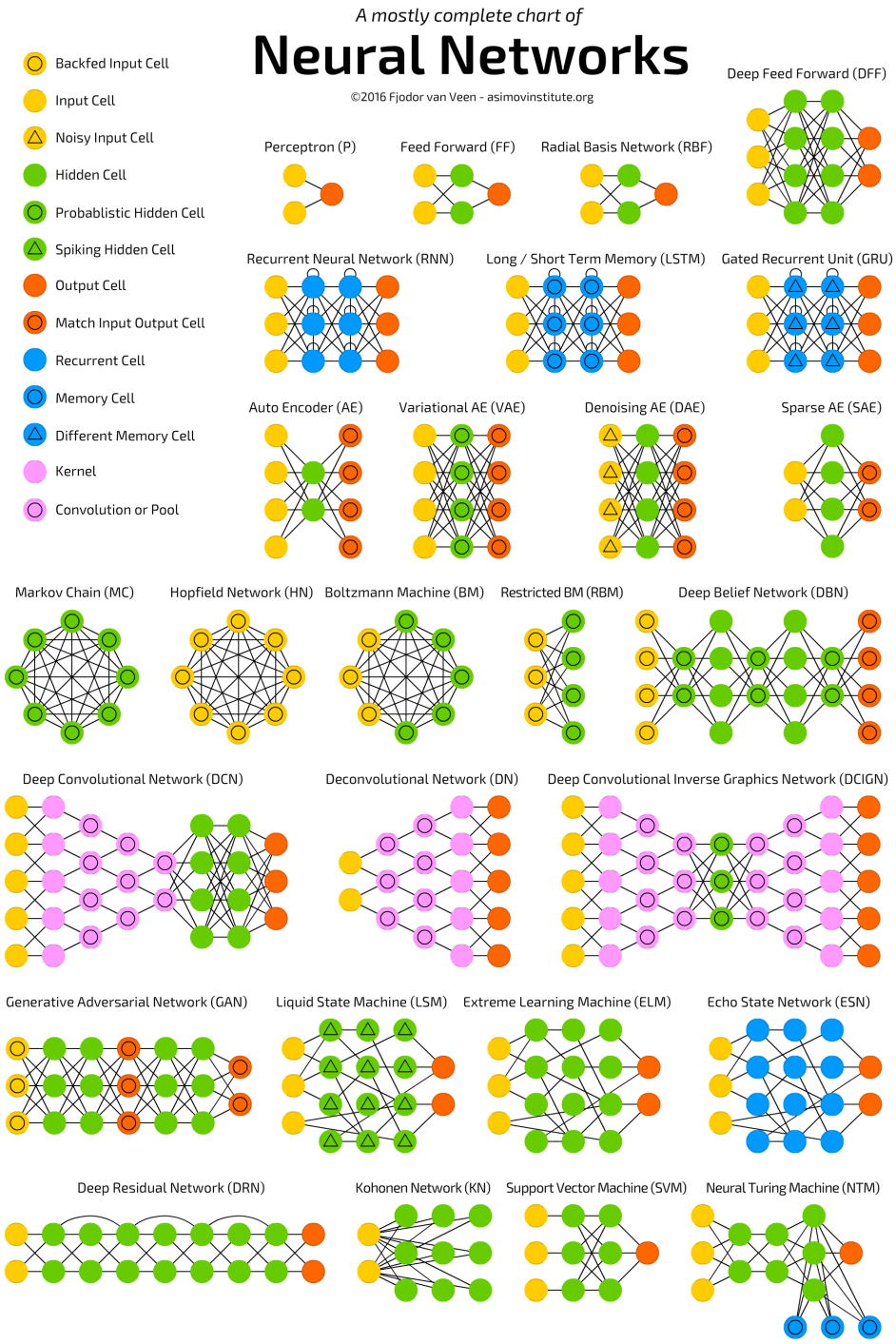


Figure 2.5: An overview on Neural Network Topologies. *Source:* http://www.asimovinstitute.org/neural-network-zoo/

2.2.4.1 Ensemble

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Unlike a statistical ensemble in statistical mechanics, which is usually infinite, a machine learning ensemble refers only to a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives [Wikd; Mur12]. For example, we can run three different algorithms, in separate, and, with the given results, run on top of them an ensemble method. There are various techniques to the choice of the best or the better model (by vote, by average, by weights, *etc*).

2.2.4.2 Hyper-parameter Tuning

In the domain of ML, *Hyper-parameter Tuning* is a task of meta learning. First, let us establish the difference between a Hyper-parameter Tuning and a normal non-hyper model parameter.

ML models are nothing more than mathematical functions that represent the link between different aspects of data. For example, a model that "obeys" to a linear regression uses a line to represent the connection *feature-target*, being mathematically represented like:

$$w^T x = y \tag{2.2}$$

Being x the vector that represents the features of the data and y a scalar variable that represents the target (some quantity that the model "wishes" to learn to predict).

The model presented concludes that the relationship between x and y is linear. w is a weight vector that specifies the gradient of the line. This is what a model knows, i.e. what it receives as parameter, which is learned during the training period. Training a given model involves optimization procedures to determine the best model parameter that *fits* the data.

As explained by Zheng [Zhe15], there is another set of parameters known as hyper-parameters, sometimes also knowns as *nuisance parameters*. Some values must be specified outside the training procedure, such as the "network" topology, the number of layers, hidden layers size, *etc*. There is a lot of already known algorithms that have Hyper-parameter Tuning. For example, decision trees have hyper-parameters such as the desired depth and number of leaves in the tree; SVM requires setting a misclassification penalty term; kernelized SVM requires setting kernel parameters like the width for radial basis function kernels; and the list goes on. [Zhe15]

2.3 Unsupervised Learning

Unsupervised Learning is another branch of ML. Its goal is to infer a function to describe hidden structure from "unlabeled" data. Contrary to supervised learning, the model only has the information on the input data, not having corresponding output variables. The goal for this type of learning is to model the elementary structure in the data in order to learn more about it. Contrary to supervised

learning, there is no correct answers and there is no teacher. Algorithms are left to their own tasks to discover and extract the interesting structure in the data.

The main problems tackled by this branch can be grouped in:

- **Clustering.** A clustering problem consists on discovering the inherent groups in the data.
- **Association.** An association rule learning problem consists on discovering rules that are able to describe large portions of the data.

Some of the more used algorithms in unsupervised learning are: *kNN*, *Anomaly Detection*, *Hebbian Learning and Learning Latent Variable Models* such as *Expectation–Maximization Algorithm*, *Method of Moments* (Mean, Covariance) and *Blind Signal Separation Techniques* (*Principal Component Analysis*, *Independent Component Analysis*, *Non-negative Matrix Factorization*, *Singular Value Decomposition*) [Gho]. Some of the algorithms referred in Section 2.2.3 (p. 9) can also be classified as unsupervised learning algorithms (for example, DL).

Although it is a very interesting field, unsupervised learning will not be developed further more on this dissertation, because it is out of scope.

2.4 Feature Engineering

Feature Engineering is an informal topic, but one that is known and agreed to be key to success in applied ML. The features of the dataset will have direct influence in the predictive models used and on the process of achieving the expected results. The chosen features need to describe the structures inherent to the data. Better features means *flexibility*, *simpler models* and *better results*. [Bro]

2.4.1 Dimensionality Reduction

A feature is an individual measurable property of a phenomenon being observed. Using a set of features, any ML algorithm can do classification tasks. During the last years, the ML applications have expanded their features' domain from tens to hundreds and hundreds of features used. Therefore, several techniques have been and are being developed to reduce irrelevant and redundant features. This is problem commonly known as *Dimensionality Reduction*. [Wikc; Mur12]

One of the approaches to Dimensionality Reduction is *Feature Selection* (or, in other words, variable/feature elimination), that aids to understand the data, reducing computation requirement and improving the predictor performance.

The focus of Feature Selection is to select a sub-collection of variables from the input data which can efficiently describe the data, while, at same time, trying to reduce the effects from noise or irrelevant variables; and still produce good prediction results. The dependent variables do not provide any extra information about the classes on the data and that is a noisy factor to the prediction. Ergo, the total information contained on the data set can be obtained from less and unique features, which include maximum descriptive information about the classes. Hence,

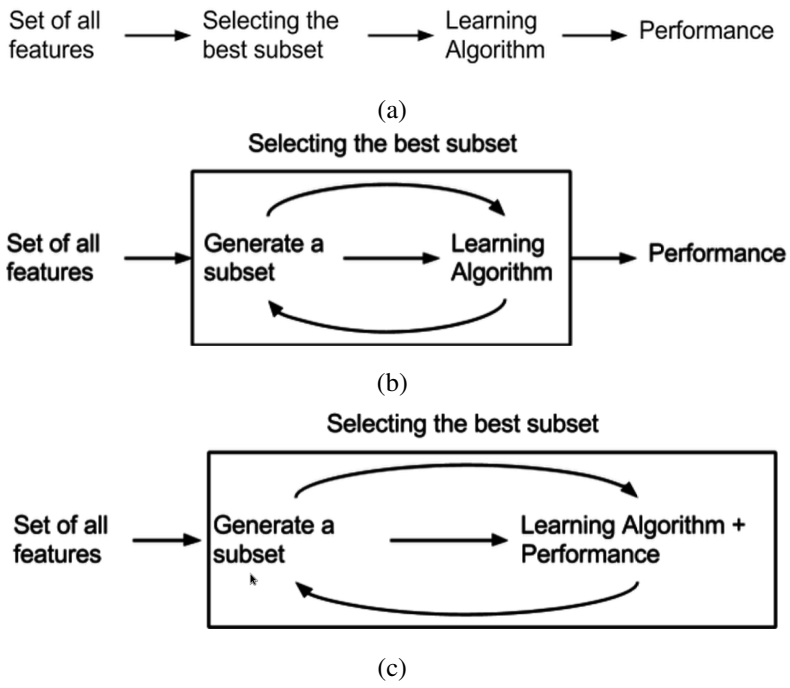


Figure 2.6: Diagram of: (a) Filter Methods; (b) Wrapper Methods; (c) Embedded Methods.

by eliminating the redundant variables, the quantity of data can be reduced, which can lead to an improvement in the classification performance. [Wikg; Mur12]

To remove an irrelevant feature, a FS criterion is required. Such criterion can measure the relevance of each feature with the output labels. From a ML point of view, if a system uses irrelevant features, it will use this information for new data leading to poor generalization. Removing irrelevant variables should not be compared with other dimension reduction methods such as PCA (*cf.* § 2.4.1.4, p. 18), since good features can be independent from the rest of the data. Feature elimination does not create new features; it uses the input features itself to reduce the number of the used ones. [CS14]

2.4.1.1 Filter Methods

Filter Methods select variables, independently of the model. They are based on general features, like the correlation with the variable to predict. Filter methods suppress the least interesting variables. The other variables will be part of a classification or a regression model used to classify or to predict data. These methods are particularly effective in terms of computational time, and exhibit increased robustness to overfitting.

Filter Methods use, as Feature Selection criterion, variable ranking techniques. Ranking methods are commonly used because of their simplicity a good success on practical applications. A proper ranking criterion is used to score the features and a threshold is used to remove the ones below it. Ranking methods are considered filter methods because they are applied before the classification process, to filter out the less relevant features.

However, filter methods tend to select redundant variables because they do not consider possible relationships between them. [CS14]

2.4.1.2 Wrapper Methods

Wrapper Methods use the predictor as a black box and its performance as the objective function to evaluate the variable subset. This techniques evaluate subsets of variables which allows, unlike filter approaches, to detect the possible interactions between variables. The two main *disadvantages* of these methods are:

- The **increased overfitting** risk when the number of observations is insufficient;
- The significant **computational time** needed when the number of variables is too large (a dataset with thousands of variables).

As defined by Chandrashekar and Sahin [CS14], the Wrapper methods are classified as SSA and HSA. The SSA start with an empty set and add features until the maximum objective function is obtained. In order to speed up the selection, a criteria is chosen which incrementally increases the objective function until the maximum is reached with the minimum number of features. HSA evaluate different subsets to optimize the objective function.

2.4.1.3 Embedded Methods

Embedded methods try to combine the advantages of both filter and wrapper methods. A learning algorithm takes advantage of its own variable selection process and performs feature selection and classification simultaneously. [Wikg; Mur12]

An example of an embedded method is the *L1 Regularization*. L1 (or LASSO) regression for generalized linear models can be understood as adding a penalty against complexity to reduce the degree of overfitting or variance of a model by adding more bias. [Rasb]

2.4.1.4 Principal Component Analysis

PCA is a useful technique when handling and managing large datasets. In some research fields, the collection of data acquired has a large number, usually more than thousands, of dimensions. Therefore, manipulating data of this dimension is not desirable, due to the practical considerations like memory and CPU time. Nevertheless, just randomly ignore its dimensions cannot be done either; it can lead to losses of important information. PCA is a common method used to manage this exchange. The idea is to, somehow, select the most important directions and keep those, while the ones that contribute mostly for noise are discarded.

PCA itself is not selecting some characteristics and discarding the others. Instead, it constructs some new characteristics that turn out to summarize our list of data. Of course these new characteristics are constructed using the old ones. In fact, PCA finds the best possible characteristics,

the ones that summarize the list of data as well as only possible (among all conceivable linear combinations). This is why it is so useful.

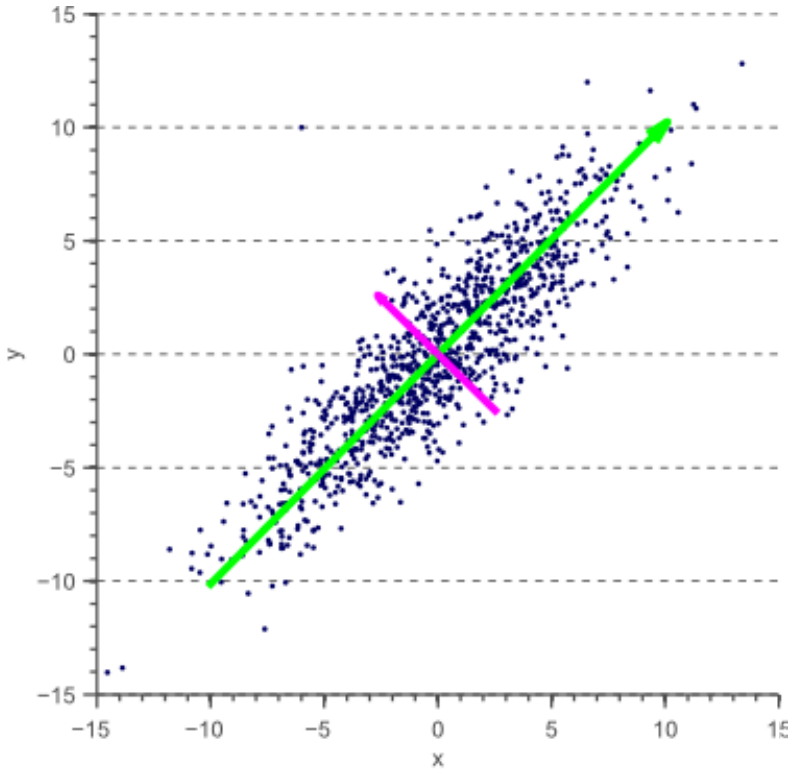


Figure 2.7: Principal Component Analysis Example, with Two Dimensions.

2.4.1.5 Autoencoders

An *autoencoder* (or autoassociator) is an ANN used for unsupervised learning of efficient codings. The aim of autoencoders is to learn a representation (encoding) for a set of data, typically for the purpose of Dimensionality Reduction. Recently, the autoencoder concept has become more widely used for learning generative models[3] of data. [Wika; Mur12]

The architecture of an autoencoder is a feed-foward, non-recurrent NN very similar to the MLP (*cf.* § 2.2.3.3, p. 11), having an input layer, an output layer and one more hidden layers connecting them. However, the output layer has the same number of nodes as the input layer. The purpose is to reconstruct its own inputs, instead of predicting the target values. Therefore, autoencoderes are unsupervised learning models. [Wika; Mur12]

[3] A *Generative Algorithm* models how the data was generated in order to categorize a signal. It asks the question: based on my generation assumptions, which category is most likely to generate this signal?

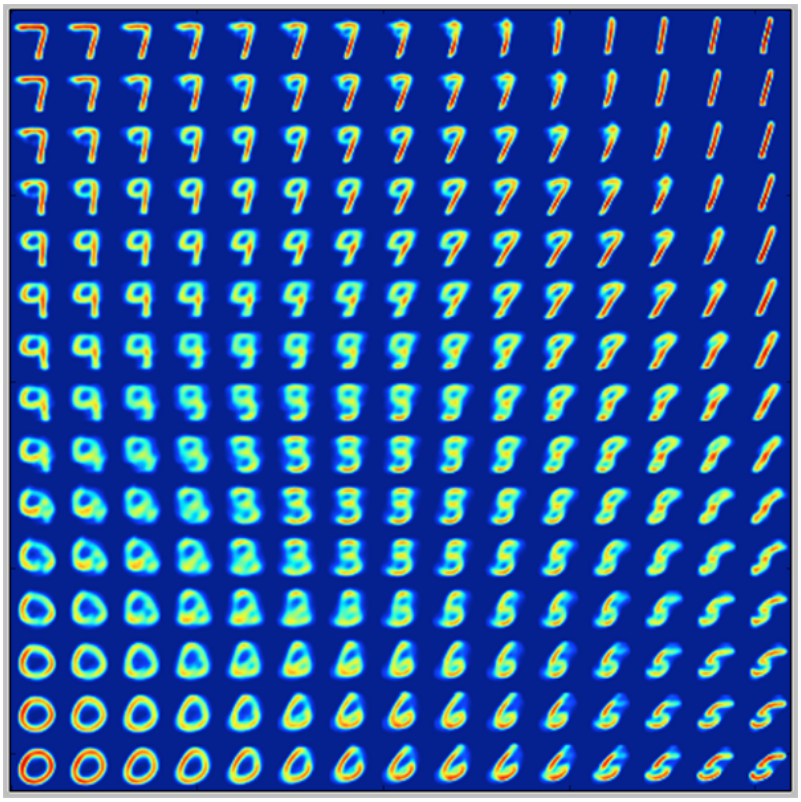


Figure 2.8: Images of a space-search of an autoencoder, programmed with *Keras* (*cf.* § 2.7.7, p. 26), that detects written numbers. Following a normal distribution, the transformations seen in the figure were achieved by fluctuating the value of the average and of the standard deviation of a statistical neuron trained for recognizing numbers. *Source:* https://blog.keras.io/building-autoencoders-in-keras.html)

Figure 2.9: Image noise removed after applying an autoencoder. This kind of autoencoder is known as *Stacked Denoising Autoencoder*. *Source:* https://blog.keras.io/building-autoencoders-in-keras.html

2.4.2 Feature Scaling

Feature Scaling is a method used to standardize the range of independent variables or features of data. It is also known as *data normalization* and is generally performed during the data preprocessing stage. The most common scaling methods are:

- **Rescaling**, which fits the range of features between 0 and 1 or -1 and 1, *ie.* $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$;

- **Standardization**, typically done by calculating standard scores; the general method is to calculate the distribution mean and standard deviation, for each feature. The next step is to subtract the mean from each feature. Then we divide the values (mean is already subtracted) of each feature by its standard deviation, *ie.* $x' = \frac{x-\bar{x}}{\sigma}$;

- **Scaling to Unit Length**, where components of a feature vector such that the complete vector become length one. This usually means dividing each component by the Euclidean length of the vector, *ie.* $x' = \frac{x}{||x||}$.

Where x' is the new scaled value of feature x. In stochastic gradient descent, feature scaling can sometimes improve the convergence speed of the algorithm. [Wikf; Rasa]

2.4.3 Deep Learning and the need of Feature Selection

As explained by LeCun et al. [LBH15], from the earliest days of pattern recognition, the goal of research has been to replace hand-engineered features with trainable multilayer networks.

Deep Learning reduces the burden of human-engineering feature selection due to their "deep" nature. Superficial layers (those that are closer to the input) will inherently be trained to recognize patterns in the input data that will be more useful to "deeper" layers. All of this happens in an unsupervised fashion way and is one of the reasons DL is becoming so predominant in the ML domain.

2.5 Statistical Comparison of Different Machine Leaning Approaches

Over the last years, studies involving several ML approaches and respective statistical comparison have been developed. This section presents some of that work.

Although none of algorithms that will be used on the thesis domain were specifically implemented for detection of MP on microscopic images, they are the ones that, on past dissertation research, had achieved the best results.

Caruana and Niculescu-Mizil [CN06] developed an empirical comparison of supervised learning models. They used the following algorithms: SVM, ANN, Logistic Regression, kNN, NB, RF, DT, *Bagged-Tree* (BAG-DT), *Boosted-Tree* (BST-DT), and *Boosted Stumps* (BST-STMP). On the first four, they scale the attributes to 0 means 1 standard deviation, applying a normalization to the features. On all the others there was no transformation on the data.

They emphasized eight performance metrics, which are divided in three different groups:

- **Threshold Metrics**, such as *Accuracy*, *F-Score* (FSC) and *Lift*;

- **Ordering/Rank Metrics**, such as *Received Operating Characteristic* (ROC), *Average Precision* (APR), and *Precision/Recall Break Even Point* (BEP);

- **Probability Metrics**, such as *Root Mean Squared Error*, (RMSE) and *Minimum Cross Entropy* (MCE).

In total, they trained about 2000 different models in each trial, for each problem. Some of the algorithms used are not designed to predict probabilities (*eg.* SVM). Therefore, they used *Calibration Methods* to ensure that the output result by all the models can be used on a statistical comparison.

They concluded that some learning method such as BST-DT, RF, BAG-DT, and SVM achieve excellent results that would be difficult to obtain just a few years back in time. The most recent methods, like feed-forward NN, have better performance and are really competitive with what's new in the area, particularly if the models do not suffer any calibration after the training phase. They also concluded that calibration methods improve drastically the performance of BST-DT, SVM, BST-STMP and NB and provide a small but distinguished improvement for RF. For NN, BAG-DT and Logistic Regression, the improvements are not significantly improved by calibration.

Williams et al. [WZA06] also made a performance comparison of five ML algorithms, applied to the domain of practical IP traffic flow classification. The author used K-FOLD CROSS VALIDATION - the dataset was divided into *k* subsets. At each time, one of the *k* subsets was used as the test set and the others *k-1* were used as the training set. They focused only in three standard metrics - ACC, *Precision* and *Recall* - *Classification Accuracy* - and added two extra metrics - *Build Time* (time, in seconds, required to train a classifier, on a given dataset) and *Classification Speed* (the number of classifications performed per second) - *Computational Performance*. The chosen algorithms were: NB discretisation (transforms features into discrete feature, distribution models are not required), NB Kernel Density Estimation (NBK), C4.5 DT, Bayesian Network and NB. For preprocessing, the author chose to apply feature reduction (*Greedy Search* and *Best First Search*) and feature selection techniques (*Consistency-based* and *Correlation-based*).

As a result, Williams et al. concluded that the application domain had some constraints associated, which limited the number and types of features to be calculated. Nevertheless, the authors evaluated the classification accuracy and the computational performance for the algorithms by using 22 features, with two additional reduced feature sets, and concluded that:

1. **Feature reduction techniques are able to greatly reduce the feature space**, while only minimally impacting classification accuracy and at the same time significantly increasing computation performance.

2. **The majority of algorithms achieved similar levels of classification accuracy given our feature space and dataset**, making differentiation of them using the already referred metrics a difficult task.

3. **Better differentiation of algorithms can be obtained by examining computational performance metrics**; it was detected NBK to have the slowest classification speed followed by NBTREE (by a considerable margin), Bayes Net, NBD and C4.5.

2.6 Learning from Images

A 2D image can be seen as a matrix of size $w \times h \times 3$, where w is the width, h the hight and 3 the color channels (RGB). In that way, if we consider all the pixels as different features, a simple color image with standard 1080p resolution provides 6220800 features! For the majority of the already referred algorithms above, this amount of features is computationally unfeasible.

Pixels are hard to be treated as independent feature. Therefore, such interpretation is not the best ML approach, given the specific images characteristics, such as *translation invariance*, *rotation*, *scale* and even *color*. [RCE+16]

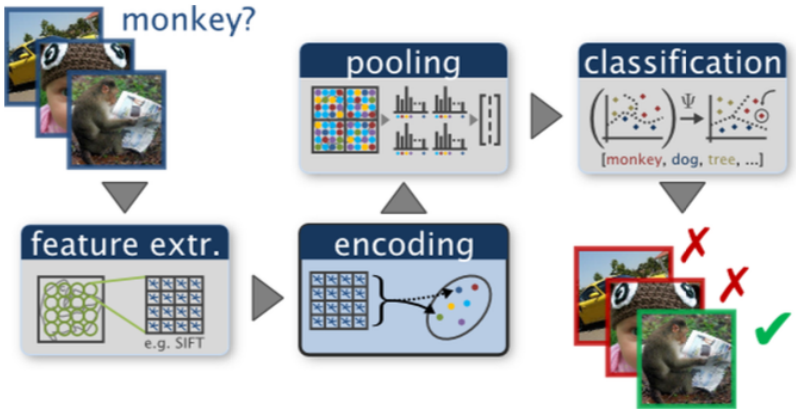


Figure 2.10: Example of the whole image encoding pipeline within an image classification task, taken from [Art15].

In this type of problems, the common approach is a preprocessing of the images, for the extraction of features, excluding pixels from being features. Those feature vary according to the given image, but here are some examples of features that are usually extracted: Histogram [TDK06; KIP+15; MAB13], Auto-correlogram [TDK06], RGB and HSV Space [PRP13], Wavelet Transformation, Shape Measurements, Gradients, Granulometry [MAB13], Texture [MAB13; EHZ11].

But, again, in DL approaches it is not need this preprocessing phase, due to the inherent feature extraction (*cf.* § 2.4.3, p. 21).

2.7 Learning Frameworks

In the Machine Learning domain, there has been a lot of progress, not just on the development of new, fastest and more accurate algorithms, but also on the development of new software and frameworks that facilitates the creation of ML models.

The following sections provide a quick overview of the most popular and used programming languages that have evolved on the data science field (Section 2.7.1 (p. 24), Section 2.7.2 (p. 24)), ML software in general (Section 2.7.3 (p. 24), Section 2.7.4 (p. 25)), and for Deep Learning, in particular (Section 2.7.5 (p. 25), Section 2.7.6 (p. 26), Section 2.7.7 (p. 26), and Section 2.7.8 (p. 27)).

2.7.1 *Python*

The *Python* programming language is establishing itself as one of the most popular languages for scientific computing. The language itself was not specifically designed with data analysis or scientific computing in mind, but thanks to its high-level interactive nature and its maturing ecosystem of scientific libraries, it is an appealing choice for algorithmic development and exploratory data analysis. Yet, as a general-purpose language, it is increasingly used not only in academic settings but also in industry. [PVG+11; Van16]

2.7.2 R

R is a language and environment for statistical computing and graphics. It provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, . . .) and graphical techniques. Being highly extensible, one of its strengths is how easily well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control. [Pro]

2.7.3 Rapidminer

Rapidminer is a visual programming environment. It helps and accelerates the creation, delivery, and maintenance of high-value data science. The RapidMiner Platform offers more functions than any other visual solution and it's open and extensible to support all your data science needs. The platform is used for business and commercial applications as well as for research, education, training, rapid prototyping, and application development and supports all steps of the data mining process including data preparation, results visualization, validation and optimization. [Min]

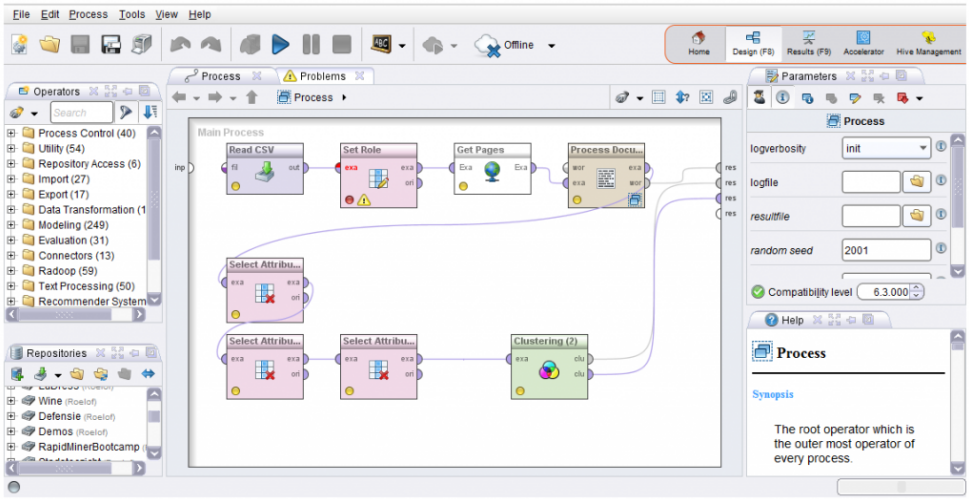


Figure 2.11: Rapidminer Platform.

2.7.4 Weka

Weka is a Data Mining software in Java; it is a collection of ML algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. [Wek]

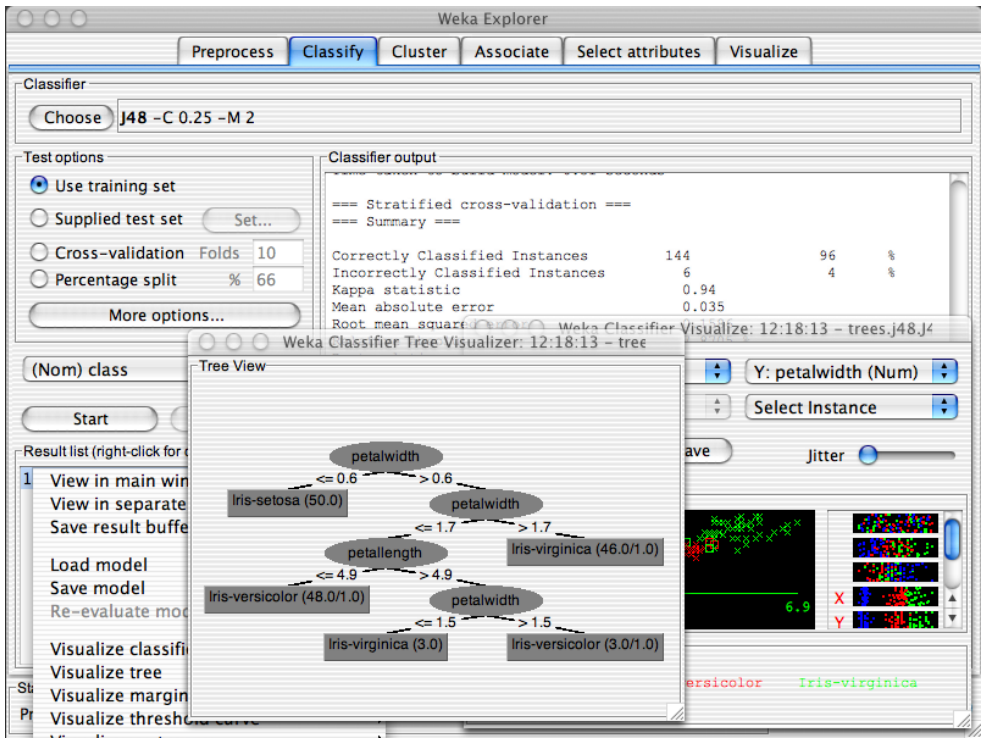


Figure 2.12: Weka Software.

2.7.5 TensorFlow

TensorFlow is a numerical programming system in which computations are represented as graphs. Nodes in the graph are called *ops* (short for operations). An op takes zero or more *Tensors*, performs some computation, and produces zero or more *Tensors*. In TensorFlow terminology, a Tensor is a typed multi-dimensional array.

TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph. TensorFlow can be used from *C*, *C++*, and *Python* programs. It is presently much easier to use the *Python* library to assemble graphs, as it provides a large set of helper functions not available in the *C* and *C++* libraries. The session libraries have equivalent functionalities for the three languages. [Ten]

2.7.6 Theano

Theano is a *Python* library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. In Theano, computations are expressed using a *NumPy*-like syntax and compiled to run efficiently on either CPU or GPU architectures. [The]

Theano has been powering large-scale computationally intensive scientific investigation since 2007. But it is also approachable enough to be used in the classroom (*University of Montreal*'s DL/ML classes). [Mon]

2.7.7 Keras

Keras is a high-level NN library, written in *Python* and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Keras provides a DL library that:

- Allows for **easy and fast prototyping**, through total modularity, minimalism, and extensibility;
- Supports both **convolutional networks and recurrent networks**, as well as combinations of the two;
- Supports **arbitrary connectivity schemes**, including multi-input and multi-output training;
- Runs **seamlessly** on CPUs and GPUs.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# split the data into a training set and a validation set
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
```

```
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

Listing 2.1: Keras Code Snippet of a Produce a Model for Word Embedding.

2.7.8 Kur

Kur is a system for quickly building and applying state-of-the-art DL models to new and exciting problems. Kur was designed to appeal to the entire ML community, from novices to veterans.

It runs over Keras and uses specification files - *.yml* files - that are simple to read and edit, meaning that it is easy to get started building sophisticated models without ever needing to code. Even so, Kur exposes a friendly and extensible API to support advanced Deep Learning architectures or workflows.

```
train:
  data:
    - mnist:
        images:
          url: "http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz"
        labels:
          url: "http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz"

model:
  - input: images
  - convolution:
      kernels: 64
      size: [2, 2]
  - activation: relu
  - flatten:
  - dense: 10
  - activation: softmax
    name: labels

include: mnist-defaults.yml
```

Listing 2.2: Kur .yml Specification File Example.

2.8 Conclusions

In this section we have seen some of the Machine Learning algorithms (in particular on Supervised Learning § 2.2 (p. 6)), converted to software tools and programming languages. When comparing

the different approaches, we have identified useful features, pros and cons, and what better would fit our work development. In Chapter 6 (p. 63), we go through the chosen technologies and its details.

Chapter 3

State of the Art in Image Detection of Malaria Parasites

In this chapter, we provide an overview on previous work that for automatic detection of malaria parasites in microscopic images. Malaria has been a word wide medical concern, as already mentioned in Section 1.1 (p. 1). In the last decade, given the severity of the disease, the complex diagnosis and the expensive equipment, research on the Artificial Intelligence and Machine Learning areas have focused on bringing faster and cheaper solutions to the most needed populations, to help improve this health issue. Malaria Parasites (MP) detection using microscopical image is usually divided into two different phases:

- *Analysis of Thick Blood Films*, where it is identified if a sample contains MP;
- *Analysis of Thin Blood Films*, where, in case a sample has been positively classified as containing MP, the stage of evolution and the species of the parasite(s) are identified.

This section will briefly review the work already developed on the application domain of this dissertation, analyzing the techniques and algorithms used.

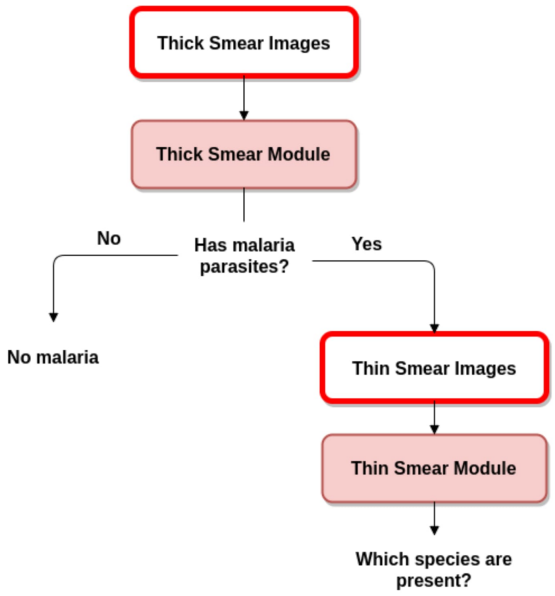


Figure 3.1: MP Detection on Thick Blood Films Process.

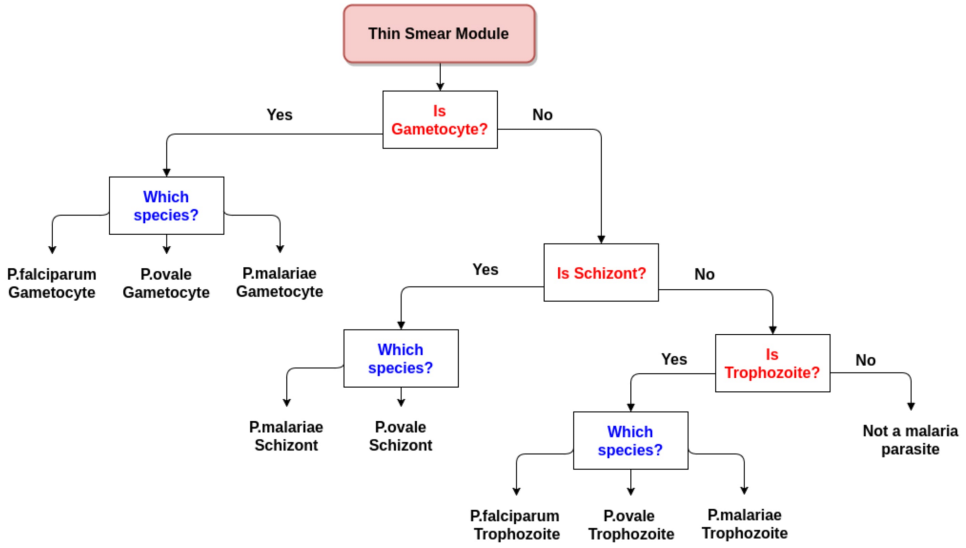


Figure 3.2: MP Detection on Thin Blood Films Process.

3.1 Thick Blood Smears

Analysis and processing of thick blood smears is the initial phase of the MP detection, as shown in Figure 3.1. Kaewkamnerd et al. [KIP+15] developed, in 2011, a detection and classification device for MP in thick blood films. The proposal was composed by two main units: *Image Acquisition Unit* and *Image Analysis Module*.

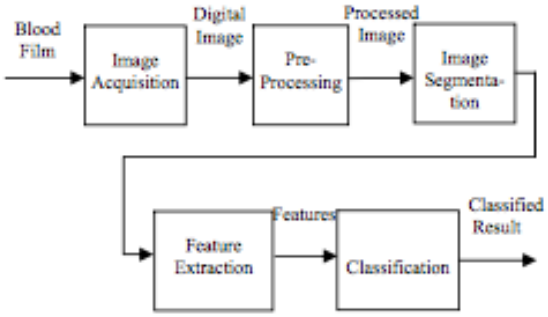


Figure 3.3: Image Analysis Module, developed by Kaewkamnerd et al. [KIP+15].

- **The first unit** was designed to be easily mounted on conventional and traditional microscopes. It controlled, automatically, the necessary movements of the microscope, in order to adjust the image, both vertically and horizontally. The microscope was augmented with a digital camera — a reflex-type *Cannon EOS 500D* — that provided the ability to capture a 1000× magnified image. Captured images were subsequently sent to the second unit installed on a personal computer.

- **The second module** detects and identifies malaria parasites on each field of the thick blood film. Before the image analysis starts, the number of fields to be analyzed was predefined and processing it required that this number was reached. Figure 3.3 shows the whole process and procedures involved on the Image Analysis Unit, namely: (i) Preprocessing, (ii) Segmentation, (iii) Feature Selection, and (iv) Classification.

The classification method used was the **chromatin size** measurement. The decision process was performed by evaluating the distribution of the referred image feature and[1]:

- If it only occupied in a range of *7—55*, the parasite was classified as *P. falciparum*;

- If the distribution was in a range of *7—150* and the chromatin size was bigger than 55 (20 percent), the parasite was classified as *P. Vivax*;

- Otherwise, it would be classified as *unknown*, since the process was only able to distinguish between two of the existing MP species.

The authors observed a performance with a success rate of, approximately, 60%. Kaewkamnerd et al. [KIP+15] concluded that this misdetection resulted from poor quality of the thick blood films and that using some other discrimination features should be used to improve the classification process developed.

Elter et al. [EHZ11] acquired 256 (*1000×1000* pixel) thick blood film images at high magnification, using *Zeiss Axio Imager Microscope* and a color *CCD* digital camera. It was taken in

[1] One can easily observe that this decision process is very akin to how a *decision tree* works.

account that all the images were capture the same lightning conditions and white balance. To form the training and the testing sets, the image set was split into two subsets (with 128 images each).

The detection algorithm had two stages: the first was focused on high detection sensitivity, with the drawback of relatively high numbers of false-positive detections; the second one used a SVM classifier to reduce the *False-Positive* detections to an acceptable level, maintaining the detection sensitivity of the first phase.

As results, the authors presented the plot on Figure 3.4 that shows the improvements on the detection of false positive, with and without the SVM algorithm. Sensitivity is the *True-Positives Rate* and concluded that, at a reasonable sensitivity of 0.97, the algorithm operates at 3.2 False-Positives per image without the false-positive reduction, and 0.8 FPI with the reduction.

Purnama et al. [PRP13] used *Genetic Programming* to identify MP on thick blood smears. The main focus of GP is automatic programming. It has been the goal of computer scientists for a number of decades. Automatic programming requires developing a computer program that can produce a desired output for a given set of inputs.

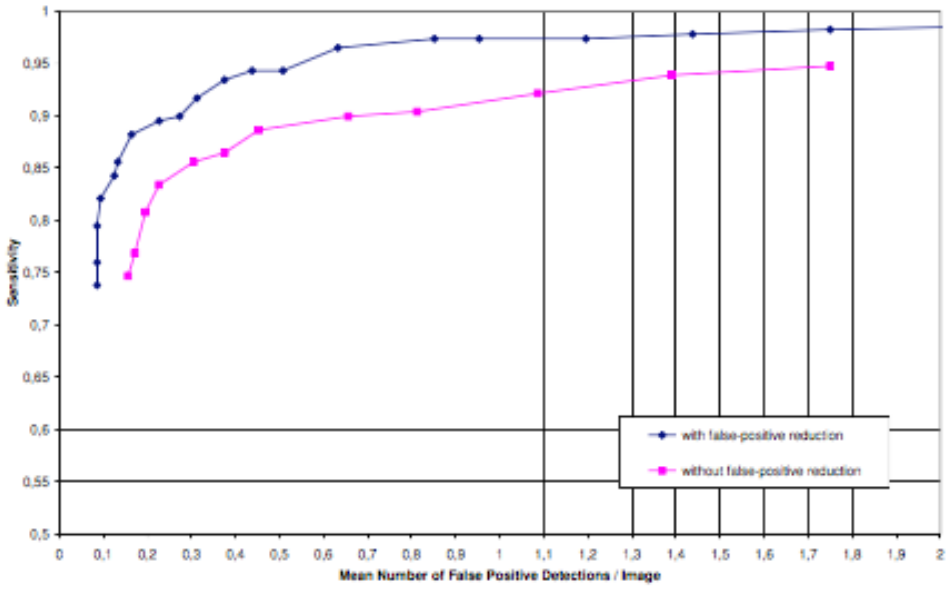


Figure 3.4: Sensitivity and False-Positive detections per image plotted as a FROC - *Free-response Receiver Operating Characteristic* - curve (with and without false-positive reduction).

The image set was composed by 180 thick blood film images which were obtained from *Dinas Kesehatan Provinsi Jawa Timur* (East Java Health Department). The images were captured using *DSLR* camera which was connected directly to the microscope. The images were cropped and resized into 64×64.

The classification result was obtained by combining four values: *True Positive* (TP - identifies correctly the samples that contain ML), *True Negative* (TN - identifies as positive samples that do

not contain ML), *False Positive* (FP - identifies as negative samples that contain ML), and *False Negative* (FN - identifies as negative samples that do not contain ML). Two experiments were made: one where the training set had 120 images and the testing set 60 and vice-versa.

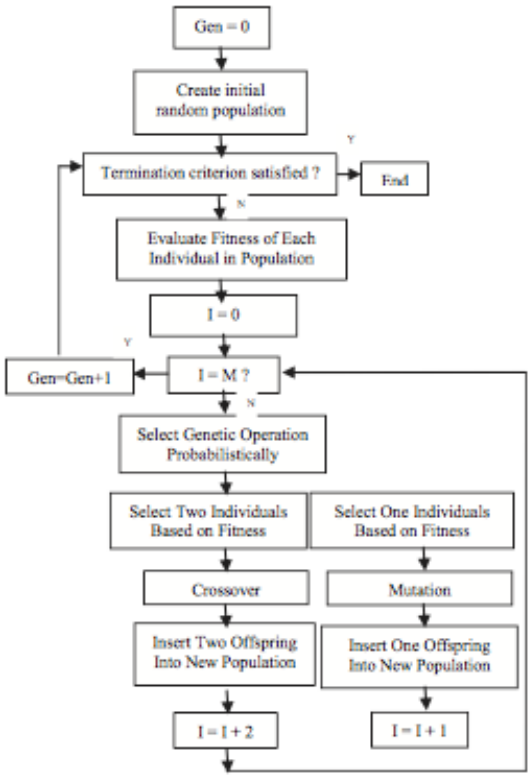


Figure 3.5: Flowchart of the GP Process.

The authors came across some great results:

- From the results of experiments on 120 training data 60 testing data of thick blood images using GP the for classification of **two classes** had 94.82% for *not parasite*, 95.00% for *parasite* and **six classes** had 88.50% for *not parasite*, 80.33% for *P. vivax thropozoit*, 73.33% for *P. vivax schizont*, 77.50% *P. vivax gametocyte*, 89.00% for *P. falciparum thropozoit*, 86.00% for *P. falciparum schizont*;

- From the results of experiments on 60 training data 120 testing data of thick blood images using GP for the classification of **two classes** had 96.17% for *not parasite*, 96.17% for *parasite* and **six classes** had 92.00% for *not parasite*, 84.17% for *P. vivax thropozoit*, 78.33% *P. vivax schizont*, 86% for *P. vivax gametocyte*, 92.50% for *P. falciparum thropozoit*, 87.5% *P. falciparum schizont*.

Yunda et al. [YAM12] developed a automated analysis method to identify, in particular, the MP specie *P. vivax*, in thick blood film images. The images were collected using a microscope image

acquisition and visualization system. This system was composed by: a microscope *Axiostar* plus, a digital camera *Canon PowerShot G6* of 7.1 MP, a personal computer and a software interface for the capture, visualization and digital image analysis. Using the descript system, a total of 163 forms of *plasmodium vivax*, *gametocyte* type, 113 forms of *plasmodium vivax* of *esquizonte* type, and 78 forms of *plasmodium falciparum*, *gametocyte* type were obtained.

From all of the forms of *plasmodium vivax*, *gametocyte* type, 85 were randomly selected to form the training group. The rest of the images were used in the classification process. The same process was done for the *plasmodium vivax*, *esquizonte* type, where 62 forms were selected for the training group, and 51 images for the classification stage. For *plasmodium falciparum*, *gametocyte* type, 42 forms were chosen for the training stage, and 36 for the classification stage.

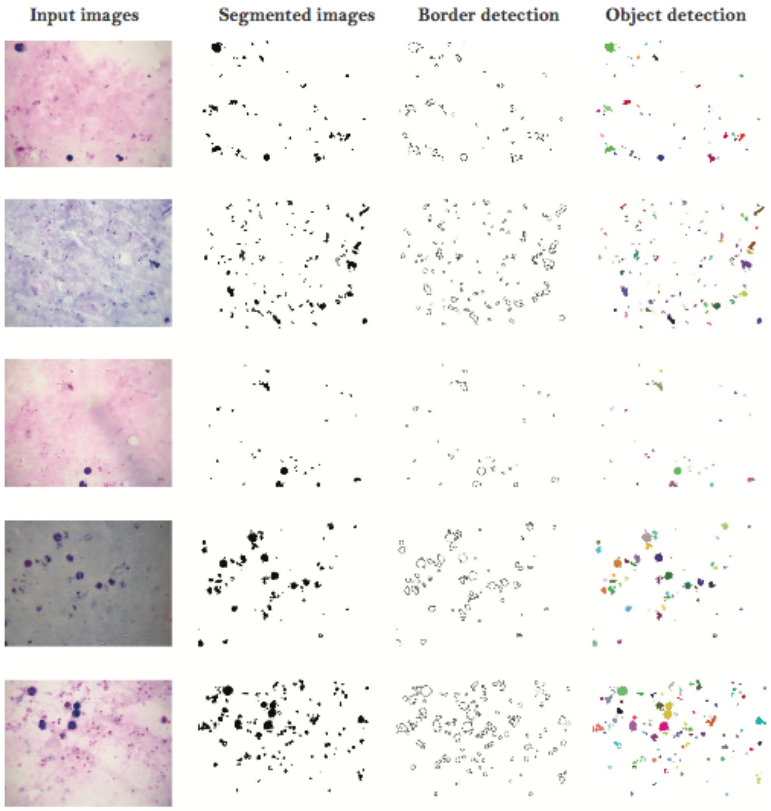


Figure 3.6: Segmentation and object detection on a set of thick blood smear images, on [YAM12].

On Figure 3.6, we have a representation of the segmentation and object detection used by the authors. For the classification phase, a multilayer Neural Network of the Perceptron was used, that was responsible for classifying *gametocytes* of *plasmodium falciparum* and *esquizontes* and *gametocytes* of the *plasmodium vivax* type.

The authors used two measures to evaluate the performance of the NN: **Sensitivity** - which measures the system's capacity to recognize at least one parasite in the thick blood smear image, without considering the type, class or the number of parasites present in the image - and **Specificity** - which takes into account the type, class and the number of parasites present in the input image. As

result, the highest percentage in the classification of the three types of parasites and the sediment class (specificity analysis) was 77.19% with a Standard Deviation of 14.17.

Quinn et al. [QAM+14] developed an automated blood smear analysis for mobile malaria diagnosis. For this, the authors design microscope that had an adapter for one of the eyepiece and a *Motic MC1000* microscope camera for the other eyepiece. The image acquisition was made with Field's stain from 133 individuals, using 1000x magnification with an oil immersion objective lens. After a pre-selection, the dataset was composed by 2703 images. Each 1024×768 image was split into 475 overlapping patches, each one of size 50×50 pixels. Given this labeled set of image patches, Quinn et al. can pose the *plasmodium* detection task as a classification problem.

To solve the problem, the authors used an Extremely Randomized Trees classifier. The classifier was trained on 75% of the labeled data (2027 images, containing 37550 patches annotated as containing parasites), and tested on the remaining 25% (676 images, containing 16312 patches annotated as containing parasites). The Receive Operating Characteristics (ROC) curve is shown in Figure 3.7, on the left, and the AUC, on the right, had the value of 0.97, indicating that the classifier was effective at distinguishing positive and negative patches.

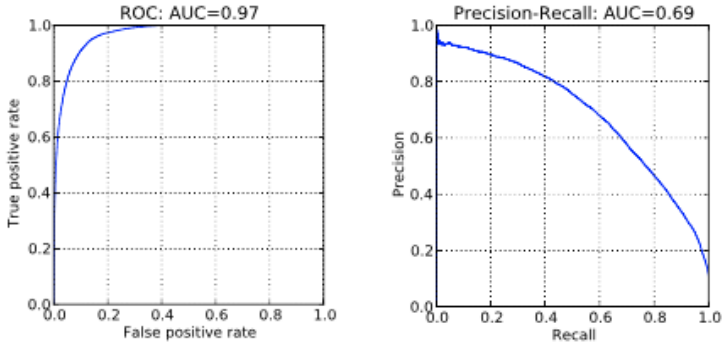


Figure 3.7: ROC Curve and Precision-Recall curves for test data. [QAM+14]

3.2 Thin Blood Smears

After the detection of MP, described in Section 3.1 (p. 30), it is necessary to know at what specie does the parasite belong and in which stage of the life cycle is it in, at the moment (Figure 3.2 (p. 30)). A thin blood sample requires a special treatment before being analyzed under the microscope. For example, the *Giemsa stain* method requires that the smear is fixed on a microscope slide in pure methanol for 30 seconds, by immersing it or by putting a few drops of methanol on the slide; then, the slide is immersed in a freshly prepared 5% Giemsa stain solution for 20–30 minutes and then flushed with tap water and left to dry. [MAB13].

Mandal et al. [MKC+10] studied and developed a segmentation of blood smear images using normalized cuts for detection of MP. *Normalized Cut* is an algorithm in which a graph is cut into two components so as to express the cost of the cut as a small fraction of the total affinity within a group. It is translate, mathematically, to the following equation:

$$NCut(A,B) = \frac{Cut(A,B)}{assoc(A,V)} + \frac{Cut(A,B)}{assoc(B,V)} \tag{3.1}$$

Where V, or, in other words, the *score of the cut*, is the weighted graph and decomposed into two components A and B. *Cut (A,B)* is the sum of weights of all edges in V that have one end in A and other in B. The *assoc(A, V)* and *assoc(B, V)* are the sum of weights of all edges weights with one end at A and B, respectively.

Additionally, the authors used segmentations techniques using the algorithm already described over the converted color space (*RGB*,*HSV* and *YCbCr* - Figure 3.8).

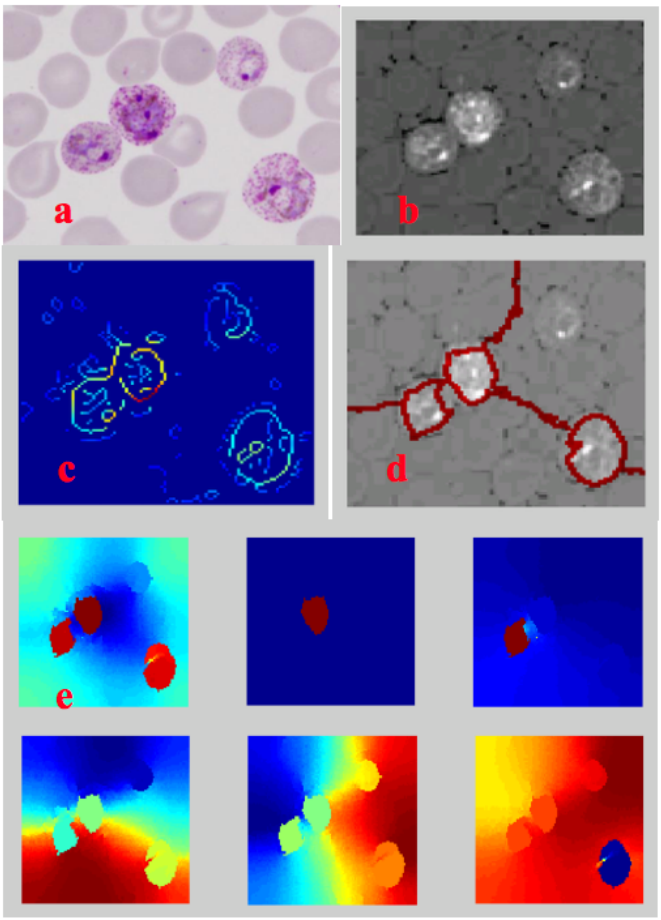


Figure 3.8: The images show segmentation in *HSV* color space; (a)Image grabbed from Microscope; (b) Brightness Image; (c) Displaying Edges; (d) Displaying the Segmentation; (e) Displaying the Eigenvectors. [MKC+10]

As conclusions, the referred paper shows a computationally efficient method for segmentation and characterization of malarial parasites from peripheral blood smear images. Its results pointed that the performance of the Normalized Cut algorithm is better in HSV color space. Nevertheless,

the authors realize that a scope of further improvement in the results can be achieved using mean shifts segmentation as a pre-processing step for this algorithm.

Zou et al. [ZCZ+10] developed a framework based on Malaria cell counting diagnosis, within a large field of view (Figure 3.9). Generally, to collect images to obtain a large field of view, it is recommended to shoot a video sequence with high-overlapped ratio between neighbor frames, which is also beneficial to create smooth mosaics. The video can be obtained through a trinocular biological microscope, which is capable to mount a video camera equipment to extract digital images by moving the sample film smoothly along the traveling stage.

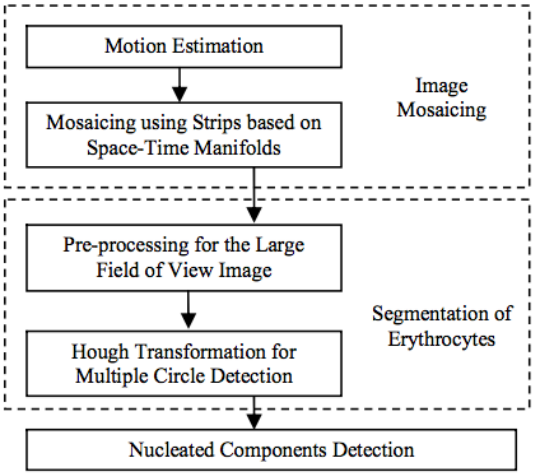


Figure 3.9: Framework for Malaria Cell Counting Diagnosis.

Image mosaicing is an essential step for improving the counting precision in the proposed scheme [ZCZ+10]. It can solve the limitation of the FOV. The performance of image mosaicing algorithm will affect the following steps directly. Taking into account the dense image capturing and the motion mode of the sample film in the application, the manifold-based image mosaicing technique is more appropriate to solve the problems for the referred conditions. Since seldom motion and calibration information is known from an external device, the motion parameters were computed between neighboring frames by using phase correlation method.

The segmentation methods implemented had to detect the *Red Blood Cells* on the digital images, since the MP allocate only on this type of cells. An improved Circle Hough Transform was used for detecting RBCs in consideration of the feature that blood cells belong to circular patterns. Compared with other segmentation algorithms, for blood cells, Circle Hough Transform is one of the optimal selections for *erythrocyte* detection due to its efficiency and robustness.

Table 3.1 summarizes the results of this framework:

Table 3.1: Results Obtained in [ZCZ+10].

Statistical Result	**Number of RBC**	**Number of Inferred RBC**	**% of Parasitemia**
Large *FOV*	31	4	12.9%
Regular *FOV*	29	4	13.8%
Large *FOV*	72	14	19.4%
Regular *FOV*	69	14	20.3%

With these, the authors concluded that a of malaria diagnose within large FOV can reduce the error of the total number of RBCs, which is caused by ignoring the cells at the borders of images, and improve the statistic accuracy for the percentage of parasitaemia.

Makkapati and Rao [MR09] developed a segmentation model to detect MP in peripheral blood smears. The models follows the segmentation heuristic of the Zou et al. [ZCZ+10]: it separates the RBC and verifies if a malaria parasite is infecting them. The twist is that the segmentation method was developed by the authors and it is color-based method. Chromatin dots are more distinguishable from the RBC than cytoplasm which is a delicate structure that shows great variability. The input images were obtained in RGB color space. Since the RGB color space is not intuitive for processing, Makkapati and Rao chose the HSV color space. HSV stands for *Hue*, *Saturation* and *Value*. HSV separates the image intensity from color information. In other words, this color space has very similar visual perception to human perception, therefore it is the most appropriated one to to recognize areas of distinct colors.

The authors concluded that the method works in HSV space and is dynamic in the sense that relevant thresholds are determined from the statistics of the given image rather than keeping them fixed for all images. The performance of the scheme was evaluated using images with color variability.

Kareem et al. [KMK11], similarly to the approaches referred before, developed a method to count the amount of RBC in a thin blood film. The authors used the annular ring ratio method, that:

- Converts the image to *grayscale*;
- Dilates the image using a ring shaped structuring element;
- Erodes the image using a disk shaped structuring element;
- Converts the closed image to a ratio transformed image by calculating the ratio of average intensities of the annular concentric ring structuring element to disk shaped structuring element masked over the image;
- Calculates the peak intensities of the ratio transformed image;
- Maps the peaks on to the corresponding coordinates, which is actually the center of each RBC.

The method has been implemented in the *Matlab* environment, for its flexibility and speed of prototyping the image processing algorithms. The authors affirm that the advantage of the proposed method is that it does not need any initial pre-processing of the images, and works a wide diversity of contrast and resolution with comparable accuracy and thus avoids unneeded complexity.

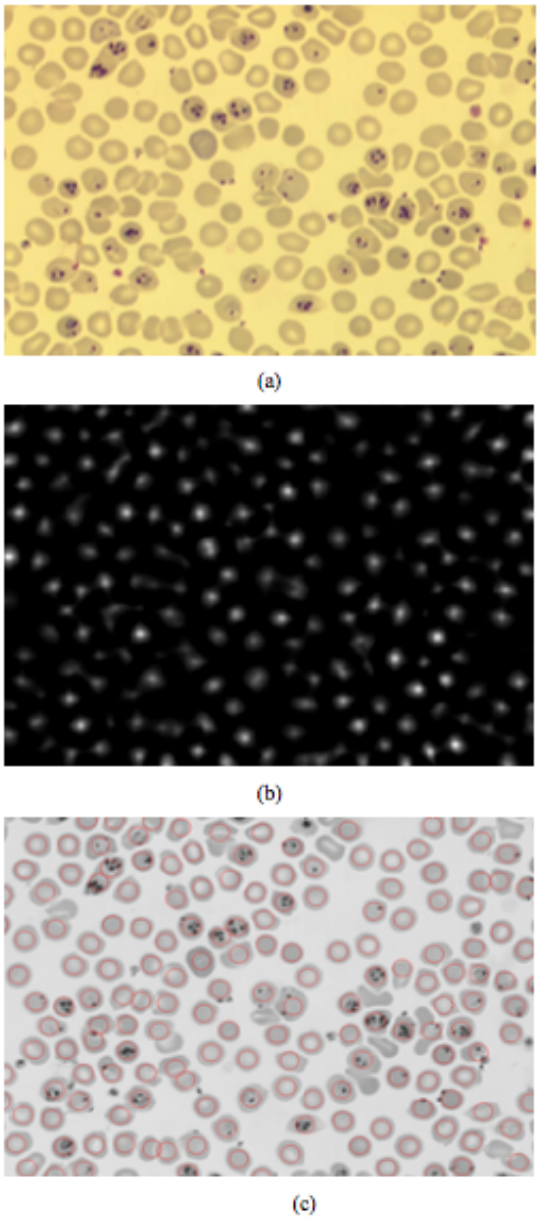


Figure 3.10: Ratio method applied on an infected image ([KMK11])- (a) Original color image; (b) Ratio transformed image; (c) Resultant RBC marked image.

Further work has been done on this domain, namely on unsupervised MP detection in blood smears images [FXL+11; KPL+14], on automated identification of MP on microscopical images [ANP+11; KKM12], on MP detection by a color image approach [MAB13; AMM13] and on MP

detection using improvements over already known algorithms [GDC+11].

3.3 Conclusions

In this chapter, we described some work done on the same application domain of this dissertation, found on the literature. Although one can find many references on MP detection on microscopical images, we did not discover any approach that considers different classification and different feature selection algorithms comparison, when applying Machine Learning techniques.

Chapter 4

Thesis Proposal

In this chapter, we first provide a more detailed study on the dataset on which we will be basing upon our work, namely (i) how the it was extracted, (ii) how the problem was approached in order to train a classifier, and (iii) their results. Using it as a baseline for our work, we will then draft our main research questions in Section 4.2 (p. 44).

4.1 Baseline Study

Rosado et al. [RCE+16] developed a methodology in *C++*, using the *OpenCV* library, to detect candidates in microscopical images, divided into three major blocks:

1. *Optical Circle* detection;
2. *White Blood Cells* (WBC) detection;
3. *Trophozoites* detection.

4.1.1 Dataset Extraction

For the first phase, the image was firstly converted to grayscale and a median filter with large window factor was applied to eliminate the inner structures. The optical circle segmentation was performed using the *Otsu's Method*. In computer vision and image processing, the *Otsu's Method* - named after Nobuyuki Otsu - is used to automatically perform clustering-based image thresholding, or, the reduction of a graylevel image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared

distances is constant), so that their inter-class variance is maximal. [Ots79] The remaining inner structures inside the optical circle were removed using a flood fill algorithm.

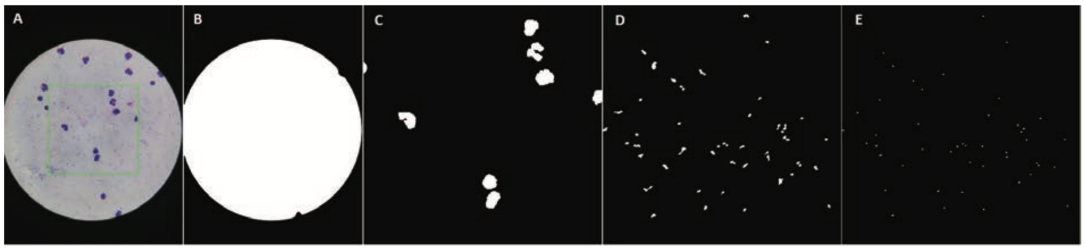


Figure 4.1: Segmentation Results: (a) Original image with *Region Of Interest* (ROI) highlighted in green; (b) Optical Circle Segmentation Mask; (c) WBC Candidates Segmentation Mask of ROI; (d) Trophozoites Candidates Segmentation Mask of ROI; (e) Chromatin Dots Candidates Segmentation mask of ROI.

A dataset of microscopic images acquired from 6 different thick blood smears infected with *P.falciparum* was used for development and testing of the proposed approach. The images were acquired using the *µSmart Scope* prototype [ROV+17] (a cheap alternative to the current microscopes, that can easily be adapted to a smartphone and used in the field), coupled to a smartphone, as shown in Figure 2.11 (p. 24). This gadget, developed at *Fraunhofer Portugal AICOS*, guarantees the required 1000× magnification, and the smartphone camera is used to capture images.

Figure 4.2: *µSmart Scope* prototype, developed at *Fraunhofer Portugal AICOS*.

WBC counts during malaria are generally characterized as being low to normal, a phenomenon that is widely thought to reflect localization of leukocytes away from the peripheral circulation and to the spleen and other marginal pools, rather than actual depletion or stasis. *Leukocytosis* - leukocyte count above the normal range - in the blood is typically reported in a fraction of cases and

may be associated with concurrent infections and/or poor prognosis. Remarkably, few published studies have compared WBC counts in malarial parasite–infected and uninfected residents of regions in which malaria is endemic, however. [MPM+05]

The second phase is divided into two main tasks: (i) *WBC Nuclei Candidates Segmentation* and (ii) *WBC Nuclei Candidates Feature Extraction and Classification*. In (i), to detect the WBC candidates, a pre-processing technique called *Mean Shift Filtering* was firstly applied, using a spatial window radius of 5 and a color window radius of 15. This algorithm is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm. [Che95] An adaptive thresholding is then applied to segment the WBC candidates. In (ii), each WBC observation was labeled according to the manual annotation, for machine learning training purposes and, for each candidate, 152 features were extracted, grouped into 3 groups: Geometry, Color, and Texture.

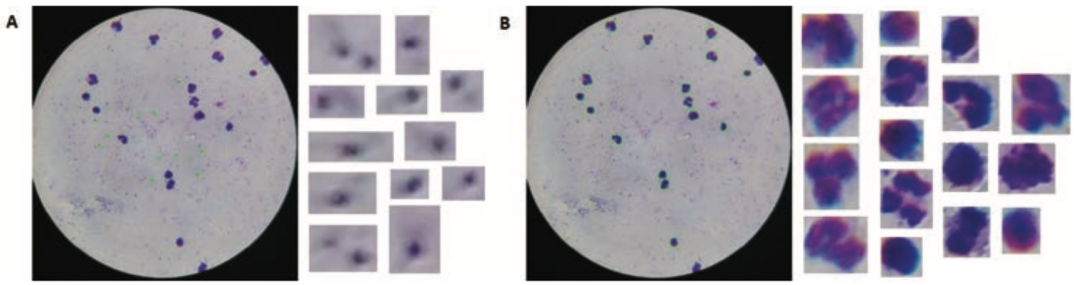


Figure 4.3: Image processing and feature extraction pipeline, developed by Rosado et al. [RCE+16]: (a) Trophozoites manual annotation; (b) White Blood Cells manual annotation.

Trophozoites are composed by a cytoplasm and one or two small chromatin dots. Using *Giemsa stain*, their cytoplasm usually stains blue and takes different shapes, from a well-defined, fine ring to forms that are irregular or bizarre, sometimes called amoeboid. The third phase refereed is composed by three different stages: (i) *Trophozoites Cytoplasm Candidates Segmentation*, (ii) *Chromatin Dots Candidates Segmentation* and (iii) *Trophozoites Candidates Feature Extraction and Classification*. In (i), the same adaptive thresholding technique applied in the WBC segmentation is also applied, with different parameters because of the parasites dimensions, in the trophozoites cytoplasm segmentation with a posterior usage of an area filtering process.

The chromatin dot is a part of the parasite nucleus, usually round in shape and stains red with *Giemsa* stain. Thus, in (ii), the algorithm consists on creating a local differences mask. The original image is blurred using a median blur filter. The small structures will disappear with the blurring process, among them the chromatin dots. The original image is then subtracted to the blurred image, and a 3-channel local differences mask is obtained.

Each trophozoite candidate will be composed by a trophozoite cytoplasm candidate and at least one chromatin dot candidate. In (iii), a total of 314 features are extracted for each trophozoite candidate: the 152 image features are also extracted for the correspondent trophozoite cytoplasm

and for the chromatin dots candidates.

At the end of the described process, a total of 314 features are extracted for each trophozoite candidate: the 152 image features referred on the second phase above, extracted for the correspondent trophozoite cytoplasm and for the chromatin dots candidates, plus 10 ratios between specific cytoplasm and chromatin features. [RCE+16]

4.1.2 Previous Machine Learning Results

A two-class SVM classifier with a *Radial Basis Function* (RBF) [1] kernel was used to create a classification model, using a grid-search approach [2] to obtain the best γ and C parameters, and a candidate region-wise 10-fold cross-validation. The same approach was applied on the second and third phases, on the classification step.

Rosado et al. [RCE+16] presents the results according to three different metrics, as detailed in Table 4.1:

- **Sensitivity**, as the percentage of candidates correctly classified as WBC/MP;
- **Specificity**, as the percentage of candidates correctly classified as Non-WBC/Non-MP;
- **Accuracy**, as the percentage of candidates correctly classified overall.

	SVM Params	TP	TN	FP	FN	Sensivity	Specificity	Accuracy
WBC	y = 0.2 C = 1.0	1817	191	74	33	98.2%	72.1%	94.9%
Trophozoites	y = 0.03 C = 0.025	223	1469	97	54	80.5%	93.8%	91.8%

Table 4.1: SVM Classification Results (TP stands for *True Positives*, TN for *True Negatives*, FP for *False Positives*, and FN for *False Negatives*).

4.2 Research Questions

Given the aforementioned methodology and results, and considering we want to use the same dataset for our work, we are now able to pose the following main research questions:

H_1. Would the application of different *Feature Selection* methods, in order to reduce the *Dimensionality* of the dataset, lead us to better classification results?

H_2. Would the application of different *Classifiers* lead us to better classification results?

[1] A RBF is a real-valued function whose value depends only on the distance from the origin, or, alternatively, on the distance from some other point c, called a center. The norm is usually Euclidean distance, although other distance functions are also possible. *Source*: https://en.wikipedia.org/wiki/Radial_basis_function

[2] Exhaustive searching through a manually specified subset of the *hyperparameter* space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set. *Source*: https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)

H_3. Would an optimization of the hyper-parameters of both the *Feature Selection* methods and used *Classifiers* lead us to better classification results?

H_4. Would the final results be statistically and significantly better than the established baseline?

In order to objectively answer the above questions, we will also need to specify exactly:

Q_1. What does one understands as a *better classification result*?

Q_2. How can one provide statistical evidence that one classifier is significantly better than another?

Hence, the main goal of this dissertation can be stated as: ***Having as basis the dataset and results obtained by Rosado et al. [RCE+16], derive a systematic method that can (i) explore a wide space of feature selection methods, classifiers, and tuning of the hyperparameters, to provide a classifier able to identify the presence of malaria in microscopical images, and (ii) provide statistical evidence of the performance of each obtained classifier, when compared to others, by relying on robust metrics.*** We will ponder over these questions during the development of the next chapter.

Chapter 5

Statistical Comparison of Different Approaches

This chapter tackles the main research questions by presenting an initial statistical analysis of the dataset (*cf.* § 5.1). We then briefly introduce the feature selection and classification algorithms (*cf.* § 5.3, p. 50) (*cf.* § 5.4, p. 51) used for this statistical study, and proceed to explain the parameter optimization function and the training/test phase for the models (*cf.* § 5.5, p. 52). Because of its relevance, we also do a brief study on the metrics robustness (*cf.* § 5.9.1, p. 56) (taken into account the models performance from the cross-validation process (*cf.* § 5.7, p. 54)) and on the logic behind the *T-Test* performed. Finally, a discussion about the achieved results is presented (*cf.* § 5.9.3, p. 58).

5.1 Preliminary Dataset Analysis

Before one starts approaching the problem of creating a machine learning approach that evidences predictive power, one should first try to have a solid grasp — and intuition — on the working dataset. For this purpose, an initial analysis focused on descriptive statistics was pursued in order to evaluate the conditions in which the data and the information are distributed.

This analysis was performed using the *Python's* PANDAS-PROFILING. This tool generates a profile report given a *Pandas DataFrame* [Pyta]. For each column/feature, the following information is presented to the user:

- **Essentials**, such as *data type*, *unique values*, and *missing values*;
- **Quantile statistics**, such as *minimum and values*, *Q1*, *median* and *Q3*, *range*, and *interquartile range*;
- **Descriptive statistics**, such as *mean*, *mode*, *standard deviation*, *sum*, *median absolute deviation*, *coefficient of variation*, *kurtosis*, and *skewness*[1];
- **Most frequent values**;
- **Histogram**, an accurate graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable).

The usage of this package will be further mentioned in Section 6.1 (p. 63).

5.1.1 Summary

The dataset contains **2651 observations**, of which **305 are labeled as infected** with malaria (our label of interest) and the remaining **2348 as not infected**. These observations contain **314 different features**, which were extracted by the image processing methodology developed by Rosado et al. [RCE+16] and described in § 4.1.1 (p. 41). **10 more features** were considered namely ratios between specific cytoplasm and chromatin features: (i) area, (ii) maximum and (iii) minimum diameter, (iv) perimeter, (v) convex hull area, (vi) bounding box area, (vii) relative difference of $C*$ and (viii) h channels mean values, (ix) difference of the coefficient of variation of $C*$ and (x) h channels. One cannot observe any kind of *null values* present in any field, which means we do not have to deal with "holes" in the information.

5.1.2 Information Gain

The tool used, PANDAS-PROFILING, also computes the features' information gain, *i.e.*, the correlation between the different features. As stated by Gennari et al. [GLF89], features are relevant if their values vary systematically with category membership. In other words, a feature is said to be *redundant* if one or more of the other features are highly correlated with it, and thus possess very little to no predictive power over the label.

[1] *Kurtosis* is a measure of whether the data is heavy-tailed or light-tailed relative to a normal distribution. *i.e.*, datasets with high kurtosis tend to have heavy tails, or outliers; datasets with low kurtosis tend to have light tails, or lack of outliers. *Skewness* is is a measure of symmetry, or more precisely, the lack of symmetry. A distribution or dataset is symmetric if it looks the same to the left and right of the center point. *Source*: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm

In order to detect if there is redundant features on the dataset, one can calculate the *Pearson's Correlation Coefficient*. This coefficient is a statistical measure of the strength of a linear relationship between paired data. In a sample, it is denoted by a measurement r, where $-1 \leq r \leq 1$. A positive value denotes positive linear correlation, and negative values denote negative linear correlation. Values in the neighborhood of 0 denotes little to no linear correlation, while the closer the value is to 1 or –1, the stronger the linear correlation. One can verbally describe the strength of the correlation using the guide that *(*Evans (1996)) suggests for the absolute value of *r*:

- **0.00 — 0.19**: Very Weak;
- **0.20 — 0.39**: Weak;
- **0.40 — 0.59**: Moderate;
- **0.60 — 0.79**: Strong;
- **0.80 — 1.00**: Very Strong.

When calculating the Pearson's correlation coefficient, it is required to hold the following data assumptions: (i) *interval or ratio level* (type of variable measurement; a interval variable is one that for which their central characteristic is that they can be measured along a continuum and they have a numerical value. A ratio variable is an interval variable, but with the added condition that 0 (zero) of the measurement indicates that there is none of that variable []), (ii) *linearly related*(a linear relationship means that you can represent the relationship between two sets of variables with a line [Tob]), and (iii) *bivariate normally distributed* (It expresses the distribution for two jointly normal random variables when their variances are equal to one and their correlation coefficient is p [IP]). In practice, the last assumption is checked by requiring both variables to be individually normally distributed. [Sta] In our working dataset, 150 features were identified as having r greater or equal than the defined threshold of 0.90.

5.1.3 Conclusions

The report seen in Appendix A, point to a case where the dataset can be considered as *extremely unbalanced*, as we have less than 12% of observations for the label of interest. We can also observe that it has an elevated number of highly-correlated features. The complete generated profile can be consulted in the — rather big — Appendix A (p. 79). Some preliminary conclusions can already be draft, namely: (i) our performance metric should take into consideration the extreme unbalance, and (ii) many features can be safely discarded with possible direct improvements on the performance of the classifiers, particularly those that assume *i.i.d*, such as Naïve Bayes.

5.2 Rationale

At the starting line, it is necessary to deliberate and design the approach, the plan. *What do we aim to discover?* is the basis question. We need to select the best combination between a Feature

Selection Algorithm (*cf.* § 5.3) and a Classifier (*cf.* § 5.4, p. 51). But they both can be optimized by getting their *hyper-parameters* tuned (*cf.* § 5.5, p. 52). For that reason, we select, on a first phase, the best feature selectors based on a "sample" using a handful of classifiers, and afterwards we select the best classifiers, using the obtained results from the selectors study. In every step, it is crucial to guarantee that all combinations are statistically different. For this purpose, we choose to rely on a statistical hypothesis test - § 5.9.3 (p. 58). Figure Figure 5.1 represents the workflow's processing line, with all the options mentioned in Section 6.3 (p. 65).

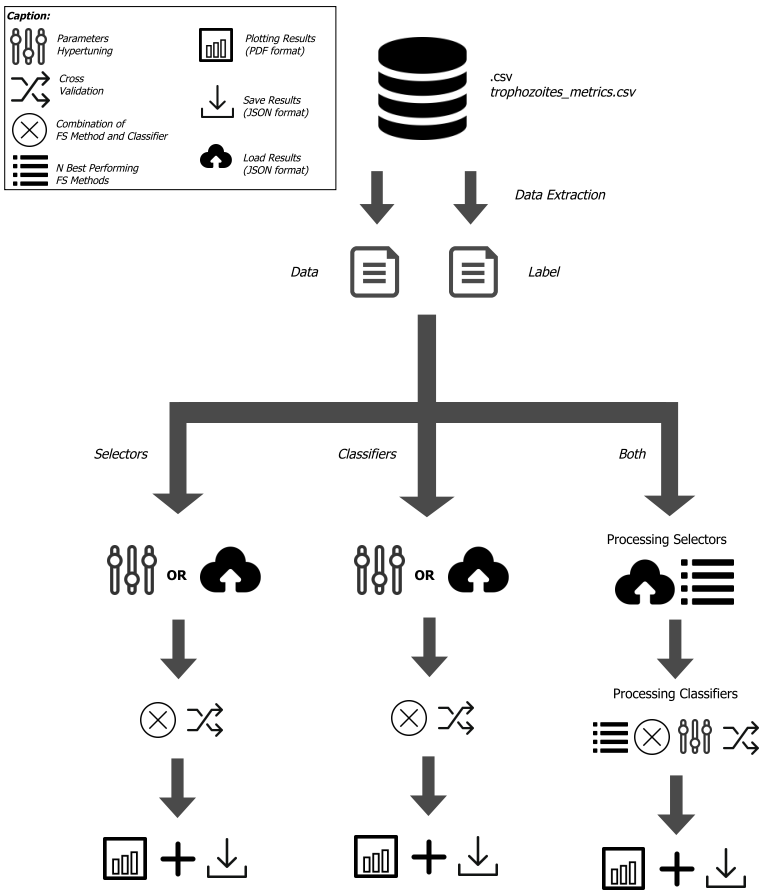


Figure 5.1: Pipeline of our approach, supported by the framework developed in Chapter 6 (p. 63).

5.3 Feature Selection Algorithms

As mentioned in Section 2.4.1 (p. 16), feature selection is one of the approaches to deal with *Dimensionality Reduction* - the process of reducing the number of random variables under consideration, by obtaining a set of principal variables to the prediction, *i.e.* the ones that provide more information gain. One of the problems with high-dimensional datasets is that, in many cases, not all the measured variables are "important" for understanding the underlying phenomena of interest. While certain computationally expensive methods can construct predictive models with high accuracy from this data, it is still of interest in many applications to reduce the dimension of the

original data prior to any modeling. In mathematical terms, this problem can be stated as follows: *given the* p-dimensional *random variable* $x = (x_1, ..., x_p)^T$ *, find a lower dimensional representation of it,* $s = (s_1, ..., s_k)$*, with* $k \leq p$*, that captures the content in the original data, according to some criterion.* [Fod02]

Thus, to deal with the referred problem in the working dataset, four different methods of feature selection were used [Mur12]:

- **Select KBest** selects features according to the k highest scores; on this method, the features are ranked according to the function *f_classif*. This function returns a value, when it is ran by an ANOVA test or a regression analysis, to find out if the means between two populations are significantly different. It is similar to a T statistic from a *T-Test*; A *T-Test* (*cf.* § 5.9.3, p. 58) will tell you if a single variable is statistically significant and this test will tell if a group of variables are jointly significant [Toa];

- **Select Fdr** selects the p-values for an estimated *False Discovery Rate*; it is a method of conceptualizing the rate of type I errors [2] in null hypothesis testing when conducting multiple comparisons. FDR-controlling procedures are designed to control the expected proportion of *discoveries* (rejected null hypotheses) that are false (incorrect rejections). The implemented method uses the *Benjamini-Hochberg* procedure [BY01]: it controls the false discovery rate under positive dependence assumptions [Wike];

- **Variance Threshold** removes all low-variance features; this algorithm looks only at the features, not the desired outputs, and can thus be used for unsupervised learning as well. This method is a simple baseline approach to feature selection. It removes all features whose variance does not meet some threshold. By default, it removes all zero-variance features, *i.e.* features that have the same value in all samples [Pytb], as they contain zero predictive power;

- The **Dummy** selector was developed only to turn the pipeline's workflow consistent and to establish the comparison between the performances with and without feature selection. This method simply does not select any features.

5.4 Classifiers

As mentioned in Section 2.2 (p. 6), classification is considered an instance of supervised learning, *i.e.* learning where a training set of correctly identified observations is available. In this work, we perform a "binary classification", *i.e.* the task of classifying the elements of a given set into two groups on the basis of a classification rule. To this matter, eleven different classifiers algorithms were evaluated [Mur12]:

[2] A type I error is the incorrect rejection of a true null hypothesis (a "false positive"), while a type II error is incorrectly retaining a false null hypothesis (a "false negative"). [Exp]

- The **Naïve Bayes** classifier assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes [Mih];
- **K-Nearest Neighbors** (KNN) is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. This algorithm is among the simplest of all machine learning algorithms [Wiki];
- The **Random Forest**, **Decision Tree**, **Extremely Randomized Trees**, **Gradient Boosting** and **Ada Boosting** classifiers belong to the ensemble methods group. An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. Ensembles can be shown to have more flexibility in the functions they can represent [Wikd];
- The **Perceptron** is a type of linear classifier, *i.e.*, a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector [Wikk];
- The **Multilayer Perceptron** classifier is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one [Wikj];
- Given a set of training examples, each marked as belonging to one or the other of two categories, a **Support Vector Machine** training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. This type of models is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible [Wiko];
- The **Dummy** classifier is one that makes predictions using simple strategies and rules. These types of classifier were implemented to be used as comparison baselines against the other classifiers. The chosen strategies were: *Stratified* (generates random predictions by respecting the training set class distribution), *Uniform* (generates predictions uniformly at random) and *Constant* (always predicts a constant label that is provided by the user; a major motivation of this method is F_1 score, when the positive class is in the minority, which is the case) [Pytc].

5.5 Tuning the *Hyper-Parameters*

Before starting to create experiments, both type of methods were submitted to an optimization process of their parameters. This process is known as *Hyper-parameter Optimization*, the problem of choosing a set of *hyper-parameters* (when defined as), with the goal of optimizing a certain measure of the algorithm's performance on a dataset. This is a common problem in machine learning. Machine learning algorithms, from logistic regression to neural networks, depend on well tuned *hyper-parameters* to reach maximum effectiveness. Since the main goal is to find best

combination of a feature selector with a classifier, we need to fluctuate the parameters values of each one to evaluate the performance evolution, in order to find, certainly, the best data model for the studied dataset. [Mur12; Wikh]

Independently, each feature selection and classification algorithms parameters were altered:

- For **KBest**, the parameter optimized was the k (the number of features that will be selected), for $k = [10, 20, 20, ..., 100]$;
- On the **Variance Threshold** selector, the parameter optimized was the *threshold*, for *threshold* = $[0.05, 0.10, ..., 0.25]$;
- In the **KNN** classifier, the parameter optimized was the *n_neighbors*, *n_neighbors* = $[1, 2, 3, ..., 10]$;
- For most of the refereed ensemble algorithms (**Random Forest**, **Extremely Randomized Trees**, **Gradient Boosting**, and **Ada Boosting**), the parameter optimized was the *n_estimators*, *i.e.* the number of *trees*. For Random Forest, *n_estimators* = $[100, 200, 300, ..., 1000]$; For Extremely Randomized Trees, *n_estimators* = $[50, 100, 150, ..., 500]$; For Gradient Boosting and Ada Boost, *n_estimators* = $[100, 150, 200, ..., 500]$;
- For **Perceptron**, the parameter optimized was the *n_iter* - the number of passes over the training data (*aka* epochs), for *n_iter* = $[5, 10, ..., 20]$;
- In the **Multilayer Perceptron** algorithm, the parameter optimized was the tuple *hidden_layers_sizes* - the *ith* element represents the number of neurons in the *ith* hidden layer, for *hidden_layers_sizes* = $[10, 20, 30, ..., 200]$;
- For the **SVM** classifier, the parameter optimized was C, for $C = [0.025, 0.05, 0.075, ..., 1]$. It corresponds to the cost of classification. A large C gives you low bias and high variance. Low bias because the cost of misclassification is penalized; a small C gives you higher bias and lower variance;

The remaining methods, previously pointed in Section 5.3 (p. 50) and Section 5.4 (p. 51), did not have any parameter optimized.

5.6 Generating Experiments

After the *hyper-tuning* process, the experiments are generated; *Experiments* is a vector obtained by the cartesian product of all feature selection methods (and its *hyper-parameters*) with all the classification algorithms. All the experiments are different from each other. When combining both methods, the selectors transform the data, selecting the features that have higher relevance, according to their heuristic.

5.7 Cross-Validation

Typically, one uses about 80% of the data for the training set and 20% for the testing set. But if the number of training cases is small for the label of interest, which is the case, this technique runs into problems, because the model will not have enough data to train on, leading to unreliable estimate of the future performance. *Cross-validation* solves this problem: the data is split into K folds; then, for each fold, all the folds are trained but the $k^(th)$, in a round-robin fashion. We then compute the average error over all the folds and use this as a proxy for the test error [Mur12]. We use 10 folds, meaning 90% of full data is used for training (and 10% for testing) in each fold test. After this process, the whole scores vector is *stored*, in order to subsequently use it for the statistical tests (*cf.* § 5.9.3, p. 58).

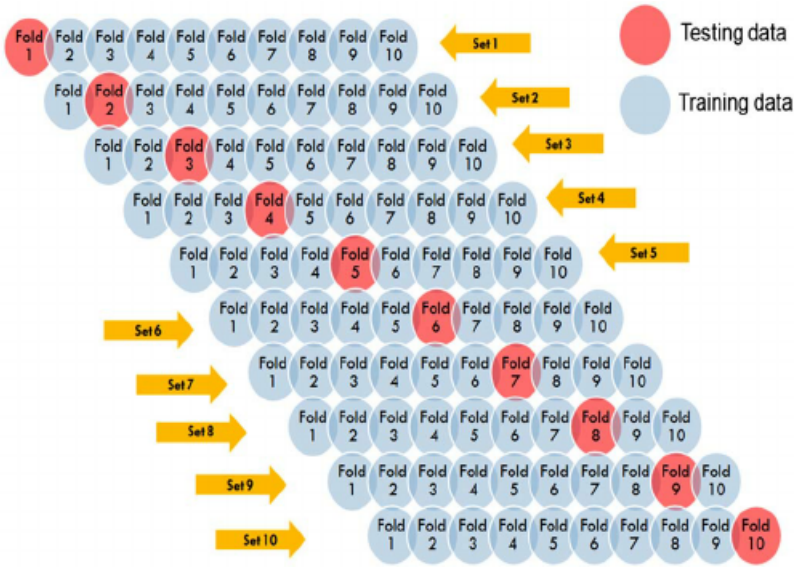


Figure 5.2: Cross-Validation, for $K = 10$ folds. *Source*: *ResearchGate*

5.8 Experimental Results

After *cross-validation*, we obtain our experimental results, our study "material". Each result has associated to it a classifier, a feature selection algorithm and an array with the *cross-validation* 10 scores. These scores are used to compute the statistical hypothesis test, further mentioned in Section 5.9.3 (p. 58). These results are visualized using charts, to better interpret the information. By having these, we can preliminary identify which seems to be the best data model, generated, since we can calculate the average performance of every experiment thanks to the storage of all the intermediate scores.

5.8.1 Sorting Results

To help visualizing the information, we ordered the experiment by their average performance, from higher to lower value. This way, one can immediately verify, specially on the *T-Test* (*cf.* § 5.8.2), if the data model is, in fact, statistically and significantly different from all the others that might be similar.

5.8.2 Visualizing Information

In order to visualize all the information generated, and to facilitate our study, we opted to create two different types of charts: the **Box Plot** and the **Heatmap Matrix**.

The Box Plot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes, indicating variability outside the upper and lower quartiles. Outliers may be plotted as individual points. Box plots are *non-parametric*: they display variation in samples of a statistical population without making any assumptions of the underlying statistical distribution. The spacings between the different parts of the box indicate the degree of dispersion and skewness in the data, and show outliers. Box plots can be drawn either horizontally or vertically. In this plot, one can find the average performance of an experiment, given a certain metric, with the respective standard deviations and outlier values. [Wikb]

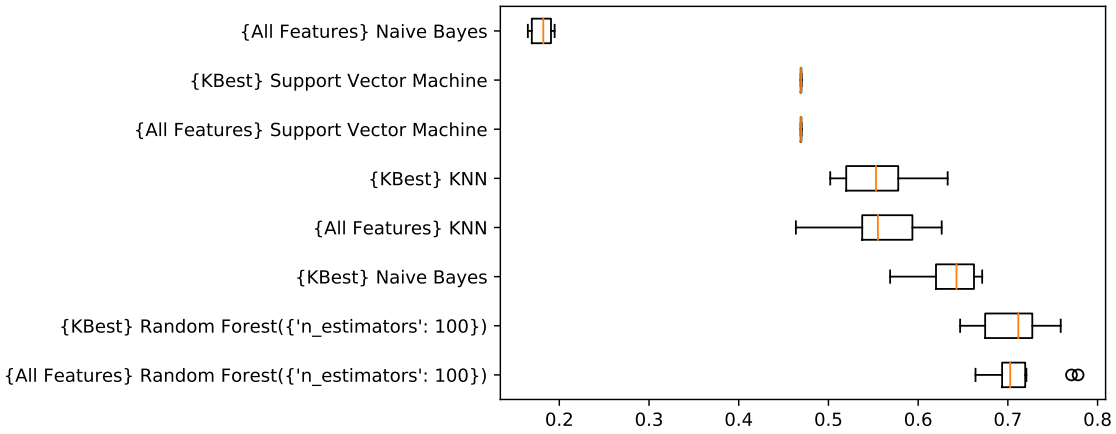


Figure 5.3: An example of a Box Plot, showing the scores of several classifiers. Inside "{}" are the Feature Selection methods used.

The Heatmap is a graphical representation of data where the individual values contained in a matrix are represented as colors. It is used to plot the several *T-Test* ran. For each Statistical Result created, its *p-value* is plotted as $1 - p\text{-}value$ since we want to find if the data models that are statistically different (see § 5.9.3 (p. 58)). Since the matrix is symmetrical, in relation to its diagonal, it is shown as a triangular matrix.

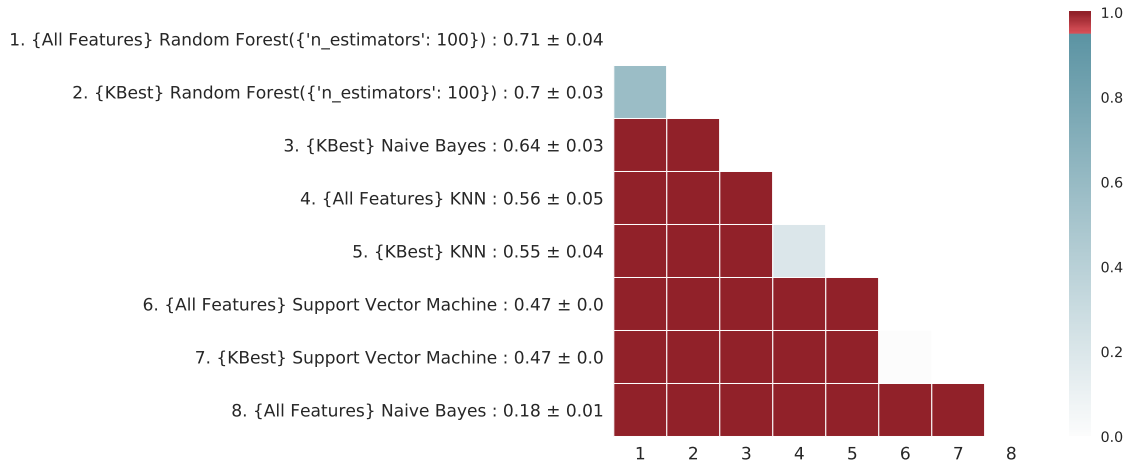


Figure 5.4: An example of a Heatmap Matrix, showing the scores of several classifiers. Inside "{}" are the Feature Selection methods used. Inside "()" are the *hyper-parameters* used.

5.9 Discussion

We have already discussed how the dataset is extremely unbalanced. In these cases, it is crucial to choose the right metric to get the most accurate results and performance, as we will no attempt to explain.

5.9.1 Metrics Overview

Among several metrics found in the literature, the most frequent ones are:

- **Accuracy**, as already referred in § 4.1.1 (p. 41), represents the percentage of instances correctly classified overall, divided by the total number of predictions made:
- **Precision** is the number of true positives (*i.e.* the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (*i.e.* the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class);
- **Recall**, also know as **Sensitivity**, is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (*i.e.* the sum of true positives and false negatives, which are items which were not labeled as belonging to the positive class but should have been);
- **Specificity**, as the true negative rate, *i.e.* the percentage of candidates correctly classified as negatives;
- **AUC** is extracted from the ROC graph; on an ROC graph, True Positives are plotted on the Y axis and False Positives on the X axis. In the ROC space, each classifier with a

given class distribution and cost matrix is represented by a point (FP, TP) on the ROC curve [HL05]. AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative');

- **F_1 score** is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score. The F_1 score can be interpreted as a weighted average of the precision and recall.

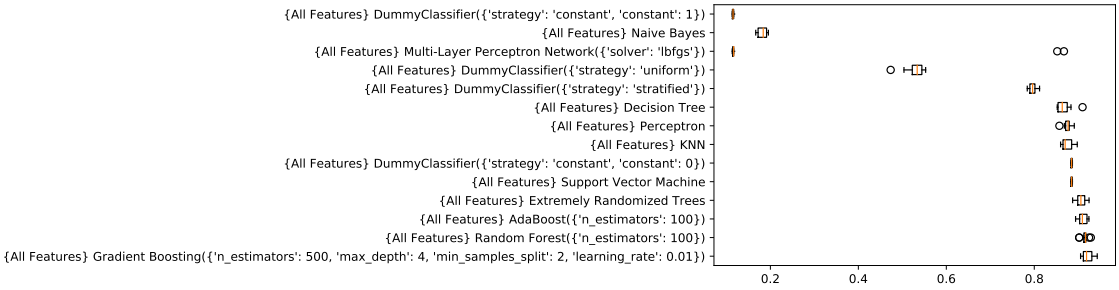
All the above reach their best value at 1 and worst at 0. Although there are other metrics, an exhaustive search was out of the scope of this dissertation.

5.9.2 Choosing the Right Metric

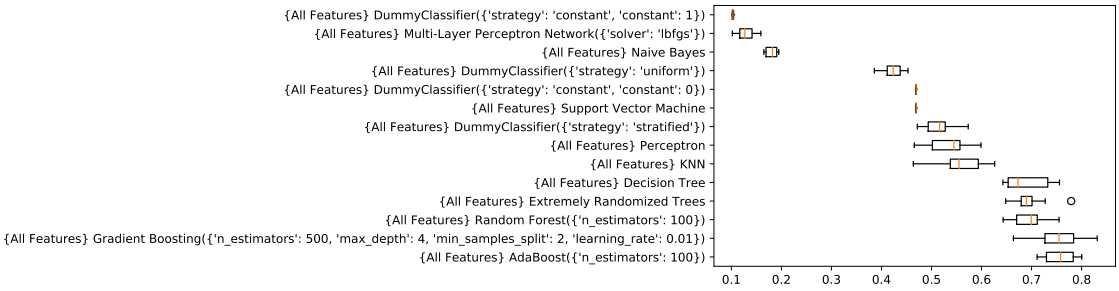
Given the fact that the dataset is extremely unbalanced, the accuracy metric can still have an elevated value if identifies all the instances that do not belong to the label of interest; or it can have a low value if, even when correctly predicts all the candidates of the label of interest. Suppose that a classifier, after the *cross-validation* process, identifies all the candidates as the label of the interest. The accuracy value would be $\frac{305}{2651} = 0.115$. In other words, the resulting model is extremely *precise*, but the *accuracy* is low.

A common metric used in the medical-related literature is **specificity**, which provides the number of negative instances correctly identified. Since our label of interest corresponds to the positive instances, this metric may have an high value, but the classifier may fail to learn how to identify positive cases, because it does not have enough data to do so.

By setting up the appropriate experimental environment, it is possible to identify which metrics are more consistent with the specific dataset under study. In this work, we analyze the relative position of the DUMMY classifiers (usually known as **ZeroR** and **OneR**, though we used some more complex (*cf.* § 5.4, p. 51) when compared to the results of other methods. As illustrated in Figure 5.4 (p. 56), one can observe that **a classifier that only predicts *Not Malaria* is more accurate then a KNN classifier, a Multi-layer Perceptron, and a Naïve Bayes**, if one is to measure **accuracy**. Different performance metrics yield different tradeoffs that are appropriate in different settings. No one metric does it all, and the metric optimized to or used for model selection does matter [CN04]. In cases where the dataset is too unbalanced, the metrics are chosen regarding the label of interest (in this case, the positive label that conveys the presence of malaria). The F_1 score is consistent with the dataset structure and proportions.



(a)



(b)

Figure 5.5: Identifying a good metric, by the position of a **dummy** classifier.

5.9.3 Hyphothesis Testing

A *statistical hypothesis*, is an hypothesis that is testable on the basis of observing a process that is modeled via a set of random variables. A *statistical hypothesis test* is a method of statistical inference: commonly, two statistical data sets are compared, or a data set obtained by sampling is compared against a synthetic data set from an idealized model. A hypothesis is proposed for the statistical relationship between the two data sets and this is compared as an alternative to an idealized null hypothesis that proposes **no relationship between two data sets**. The comparison is deemed statistically significant if the relationship between the data sets would be an unlikely realization of the null hypothesis according to a threshold probability, *i.e.* the significance level. Hypothesis tests are used in determining what outcomes of a study would lead to a rejection of the null hypothesis for a pre-specified level of significance. [Dow11]

The statistical test chosen is the *T-Test*. A *T-Test* is any statistical hypothesis test in which the test statistic follows a *Student's t-distribution*, under the null hypothesis. It is commonly used to determine if two sets of data are significantly different from each other [Dow11]. The value associated to this test is the *p-value*, *i.e.* the probability for a given statistical model that, when the null hypothesis is true, the statistical summary would be the same as or more extreme than the actual observed results; the smaller the *p-value*, the larger the significance because it tells us that the hypothesis under consideration may not adequately explain the observation.

In our case, we aim to determine if two data models are significantly distinct from each other:

- Our **alternative hypothesis** is that they are different;

- Our **null hypothesis** is that they are equal;
- The p-value corresponds to the **probability of the null hypothesis holding**, which means the two models being statistically equivalent. If, however, the $p\text{-}value \leq 0.05$, then we cannot provide evidence that the null hypothesis holds, and so we have statistical support to reject it: likely, the two models are statistically different.

Performing the *T-Test* is crucial for this study. It is not guaranteed that the best combination, on a performance level, is, in fact, the best one. One must verify if the data model with the highest performance is significantly different from all the other data models generated, by comparing every score generated for each slice of the cross-validation.

5.10 Conclusions

Experiment	Avg. F_1	σ
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 350})	0,7770	0,03
{All Features} AdaBoost({'n_estimators': 400})	0,7670	0,02
{All Features} Gradient Boosting({'n_estimators': 500})	0,7598	0,05
{Fdr} Gradient Boosting({'n_estimators': 500})	0,7568	0,04
{Variance Threshold({'threshold': 0.10)} Extremely Randomized Trees({'n_estimators': 200})	0,7293	0,03
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 100})	0,7224	0,03
{All Features} Extremely Randomized Trees({'n_estimators': 100})	0,7203	0,04
{All Features} Random Forest({'n_estimators': 400})	0,7161	0,03
{All Features} Decision Tree	0,6939	0,03
{Fdr} Decision Tree	0,6772	0,04
{KBest({'k': 100})} Naive Bayes	0,6751	0,03
{KBest({'k': 100})} KNN({'n_neighbors': 3})	0,6193	0,03
{All Features} KNN({'n_neighbors': 1})	0,5863	0,06
{All Features} Perceptron({'n_iter': 15})	0,5507	0,06
{Variance Threshold({'threshold': 0.10})} Perceptron({'n_iter': 15})	0,5507	0,06
{Variance Threshold({'threshold': 0.20)} Multi-Layer Perceptron Network({'hidden_layer_sizes': 170})	0,5366	0,05
{KBest({'k': 20})} DummyClassifier({'strategy': 'stratified'})	0,5013	0,04
{All Features} DummyClassifier({'strategy': 'stratified'})	0,5013	0,04
{All Features} Multi-Layer Perceptron Network({'hidden_layer_sizes': 180})	0,4874	0,03
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.015})	0,4695	0,00
{All Features} Support Vector Machine	0,4695	0,00
{All Features} Naive Bayes	0,1807	0,01

Table 5.1: Best results for all the implemented classifiers (with and without applying feature selection), according to F_1 score.

Just by observing Table 5.1, Figure 5.6 (p. 60) and Figure 5.7 (p. 60) we came across the following conclusions:

- **Applying feature selection methods increases the classifier's performance**. Here we have the comparison between (i) selecting a group of features, and (ii) no feature selection involved;
- **Hyper-parameter optimization increases the data model's performance**. Since we tuned the *hyper-parameters*, before creating any experiment, one can also affirm that optimizing

the classifier parameters leads to higher performance results. Appendix Chapter A (p. 79) shows the complete table, where it is possible to compare the different performance achieved by the different parameters values;

- ***T-test* proves the data model's significance**. One can rapidly verify how robust the data model is, since all it is needed is to verify if, in Figure 5.7, **the color associated to the statistical test result is red**. Figure 5.6 represents the data models statistically. In our best case scenario, it is not.

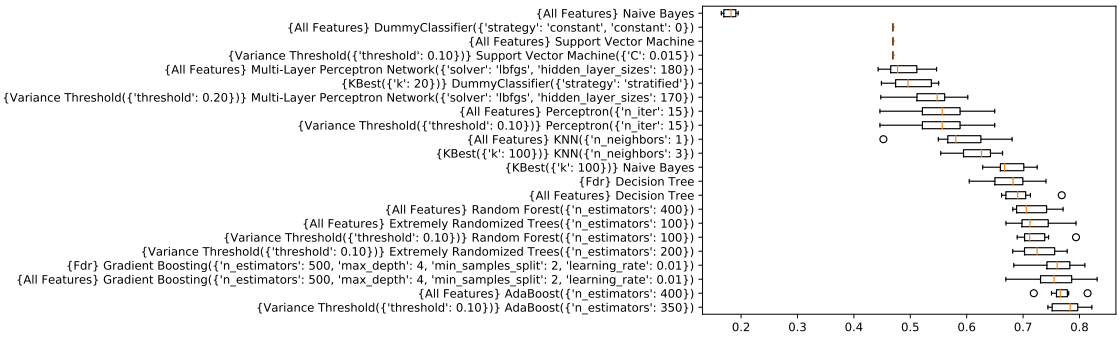


Figure 5.6: Box plot with the best results, regarding F_1 score, for all the studied classifiers.

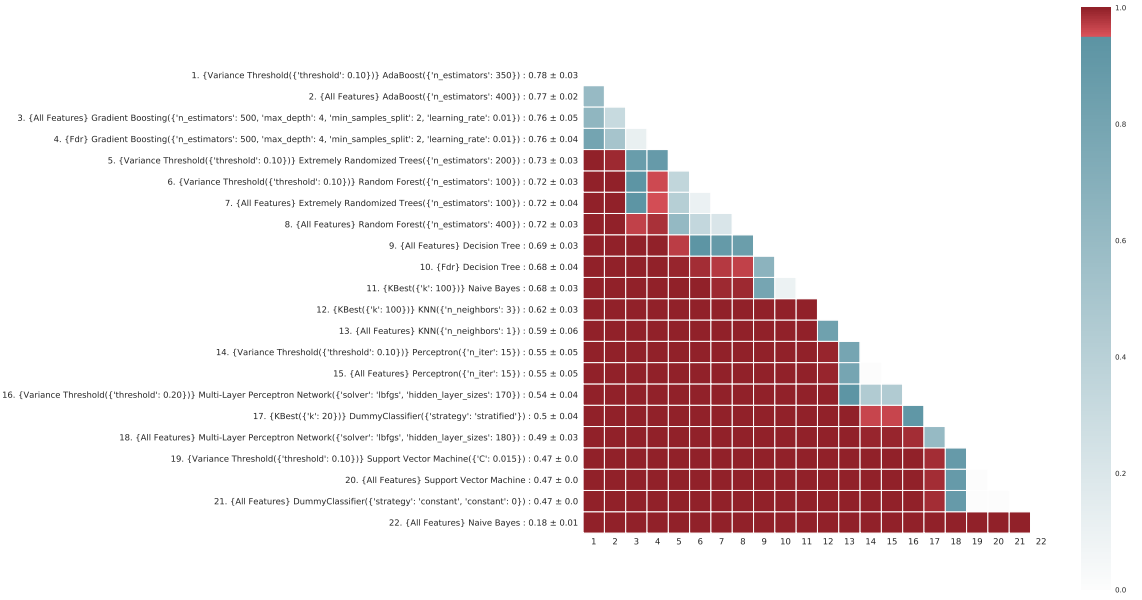


Figure 5.7: Heatmap with the best results, regarding the F_1 scoremetric, for all the studied classifiers; one can see that best result is significantly different from all the others.

Let M_1 be the first method represented in Figure 5.7, M_2 the second one, and so on. It is straightforward to conclude that the best reached result was given by M_1, an **Ada Boost** classifier with

350 estimators, combined with a **Variance Threshold** feature selector **of 0.10**, which achieved an F_1 score of 78%. One can, however, verify that the top four (4) methods, which include other *hyper-parameters* and feature selection combinations of both **Ada Boost** and **Gradient Boosting**, do not evidence statistical significance among themselves. We are then tempted to presume, according to these experiments, that they are all good classifiers with differences in the scores likely due to chance. Notwithstanding, a closer analysis also shows no statistical significance of M_3 when compared to M_4, M_5 and M_6; a scenario that do not occur with M_1 and M_2. Ergo, we can conclude, within statistical confidence, that the best performance is achieved by an **Ada Boost classifier with $\approx$350—400 estimators**, an ensemble method that, as observed, is quite robust to the *dimensionality curse* as the selection strategies do not evidence significant impact on its performance.

Chapter 6

Implementation Details

In this chapter, the main focus is toward the initial dataset study and the developed framework architecture and logic, that were essential to achieve the end results of this dissertation.

As mentioned in Section 2.7.1 (p. 24), Python has emerged over the last years as a first-class tool for scientific computing tasks, including the analysis and visualization of large datasets. The usefulness of Python for data science stems primarily from the large and active ecosystem of third-party packages like NUMPY, PANDAS, SCIPY, MATPLOTLIB, SCIKIT-LEARN, and many more that are mentioned in Section 6.1.

6.1 Libraries and Modules

The *Python* libraries [1] used on the framework's development were the following:

- SCIKIT-LEARN provides simple and efficient tools for machine learning and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting and k-means. It is built on NUMPY, SCIPY and MATPLOTLIB;
- MATPLOTLIB is a 2D plotting library which produces publication quality plots and charts in a variety of hardcopy formats and interactive environments across platforms. This library provides you "tools" to generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code;

[1] All the mentioned modules were integrated in a virtual notebook environment programming named *Jupyter Notebook* - http://jupyter.org/index.html, https://ipython.org

- PANDAS provides fast, flexible, and expressive data structures designed to make working with structured and time series data both easy and intuitive. It has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language;
- NUMPY is the fundamental package for scientific computing with *Python*. It contains, among other things, a N-dimensional array object and useful linear algebra, Fourier transform, and random number capabilities.
- PANDAS-PROFILING (already mentioned in Section 5.1 (p. 47)) generates profile reports from a PANDAS DataFrame. For each column, it presents the several statistics, in an interactive *HTML* report;
- JSONPICKLE is a library for serialization and deserialization of complex Python objects to and from *JSON*. The standard Python libraries for encoding *Python* into *JSON*, such as the STDLIB'S JSON, SIMPLEJSON, and DEMJSON, can only handle Python primitives that have a direct JSON equivalent. This library builds on top of these libraries and allows more complex data structures to be serialized to *JSON*. It is highly configurable and extendable;
- ARGPARSE makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and ARGPARSE will figure out how to parse those out of SYS.ARGV. It also automatically generates help and usage messages and issues errors, when users give the program invalid arguments.

6.2 Architecture and Design

The class *Strategy* is responsible for the instantiation of the different classification algorithms. Its methods correspond to the classifiers taken into account for the comparison, and the *hyper-tune* function (optimizes the classifiers parameters of a certain classifier - (*cf.* § 5.5, p. 52)).

Selector manages the creation of the different Feature Selection methods. Besides that, it also has the function *hyper-tune* (*cf.* § 5.5, p. 52) and the function *select*. The last one mentioned is in charge of the features selection *per se*, according to the heuristic of each method.

The *Experiment* class combines a classifier algorithm with a feature selection method, through the *combine* function, creating the different experiments. This class also includes the function *xValidation*, that applies the cross validation technique in a certain already created experiment.

The cross validation function creates a *ExperimentResult* instance. This class computes the *p-value*, through the *TTest* function, which verifies the statistical difference between two distinct data models. This function returns a *StatisticalTest* instance. The class is also responsible for the results visualization, which will be further mentioned in Section 5.8.2 (p. 55).

The following figure schematizes the framework's architecture:

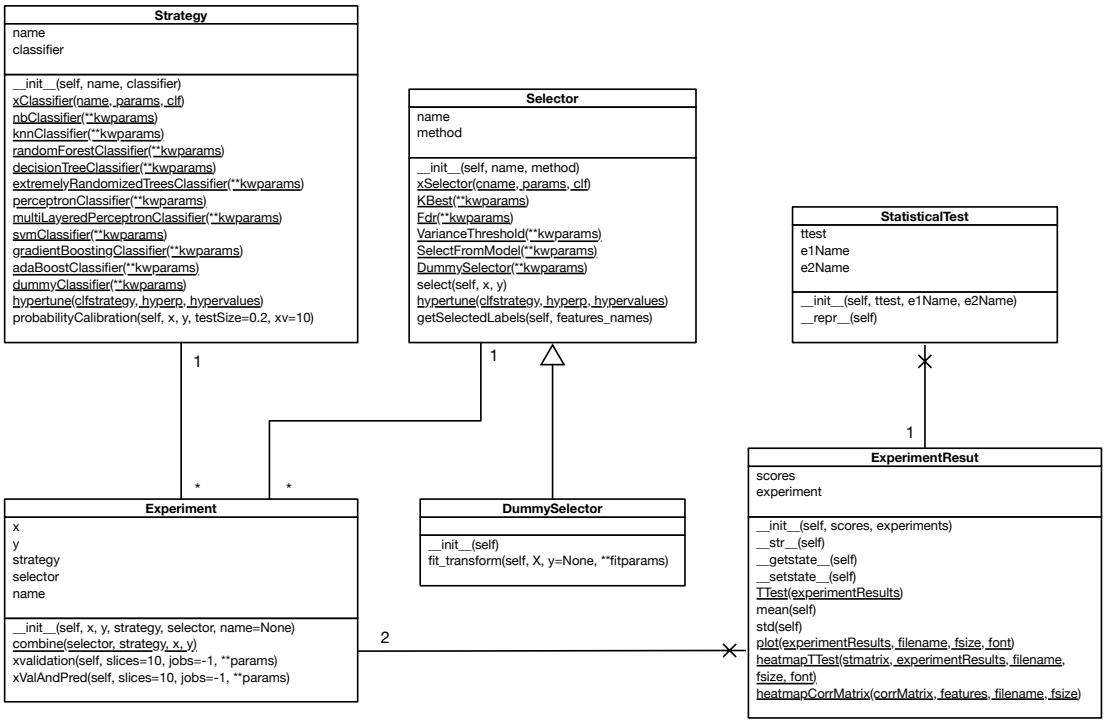


Figure 6.1: Framework Class Diagram in *UML*, with all the implemented classes and respective methods

6.3 Usage

The main usage of the tool can be summarized by invoking it with the **`-help`** argument:

```
usage: generator.py [-h] (-sel | -clf | -all) [-s SAVE] [-l LOAD]
                         [-n NSELECTORS] [--seed SEED] [-viz VISUALIZE]
                         [-mtr METRIC]
                         [filename]
```

Listing 6.1: Tool Help Menu.

This tool presents three main paths, namely: (i) feature selection, (ii) classification, and (iii) analysis. Initially, the user supplies the wanted dataset, and then he can:

`-sel, --selectors` Analyze which are the best feature selection methods, by hyper-tuning its parameters and using few classifiers with the default settings;

`-clf, --classifiers` Analyze which are the best classifiers applied to the dataset, by hyper-tuning its parameters and using some default feature selection methods - this methods were chosen after some tests and studies and these are ones that usually provide the best results;

-all Analyze which are the best combos, *i.e.*, the combination of feature selection methods and classifiers, where it is possible to define the number of the selectors that will feed the classifiers (which is 10, by default), with the optional argument **-n, --nselectors**.

After choosing a path option, but before the start of the processes, the user has also several optional arguments:

-s, --save Serializes the intermediate results to a `.json` file;

-l, --load Loads previously serialized results from a existing `.json` file;

--seed Define the seed of the random number generator for the data shuffle process;

-viz Choose the visualization output;

-mtr Select the metric used on the cross validation process.

A possible sequence of events is illustrated in Figure 6.2 (p. 67).

```
{
  "py/object": "framework.experimentresult.ExperimentResult",
  "py/state": {
      "name": "{Variance Threshold({'threshold': 0.10000000000000001})} AdaBoost({'
          n_estimators': 100})",
      "scores": [
                0.8145048814504882,
                0.7605042016806722,
                0.7668000762340386,
                0.7740350877192983,
                0.7506012800130447,
                0.7721969790003683,
                0.7272727272727273,
                0.7013574660633484,
                0.7588083416087388,
                0.8131634819532909
      ]
  }
}
```

Listing 6.2: Example of an intermediate serialization of the results.

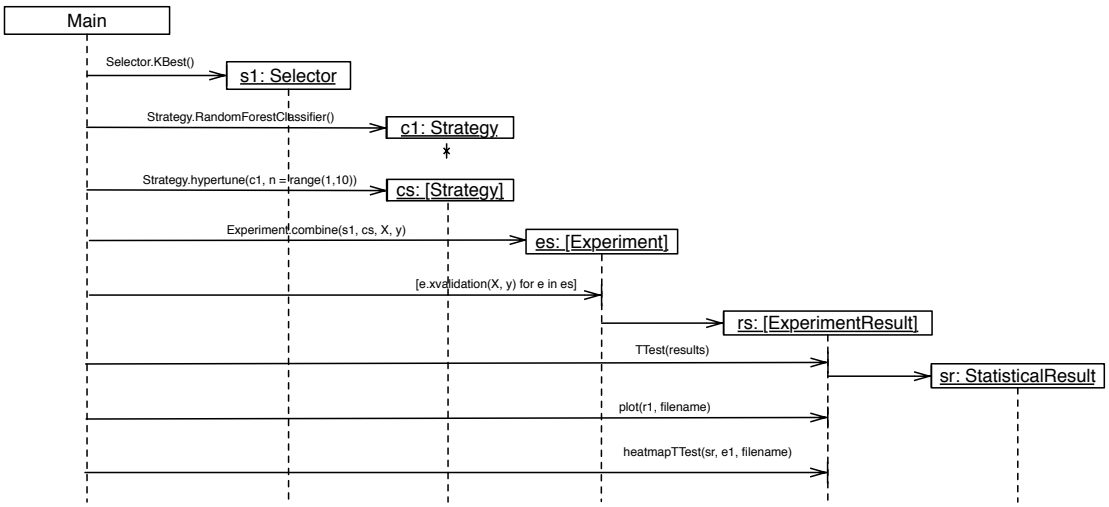


Figure 6.2: A possible UML sequence diagram, where one strategy and one feature selector are created, and the strategy is hyper-tuned over the parameter $n = [1..10]$, thus resulting in an array of Strategies. Combining those using a cartesian product yields an array of Experiments. For each experiment, we evaluate it using cross-validation and subsequently tests the results agains each other using an hypothesis test. Finally, we order the creation of some charts.

An example of usage is the following:

```
~ ./generator.py -all -s recentresults.json -viz PNG -mtr roc_auc
    trophozoits_metrics.csv
```

Listing 6.3: Tool Usage Example.

With this options, the tool will analyze which are the combination of feature selection methods and classifiers, serialize and save the results to the file *recentresults.json*, using the *roc_auc* metric, and the charts will be generated in *PNG* format.

6.4 Implementation Conclusions

We chose *Python* to develop our framework, given its design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and the many (not just, but also) Machine Learning modules, examples, and documentation already existent on this programming language.

The framework is pretty straightforward, easily used on the computer terminal, with help menu and different optional argument, so that the user can change the results, to find what he is looking for.

Chapter 7

Conclusions

In this dissertation, we started by exploring the current state-of-the-art in detecting malaria parasites in microscopic images (*cf.* Chapter 3, p. 29). We have shown how the current literature makes usage of several machine learning methods for this end (*cf.* Chapter 2, p. 5). In particular, we considered the work of Rosado et al. [RCE+16] as the baseline for our research, and attempted to provide evidence to a set of research questions (*cf.* Chapter 4, p. 41).

7.1 Summary of Research Questions

Before discussing the author's fundamental research questions, we have previously pondered over how to objectively measure an improvement over the baseline in § 4.2 (p. 44). Particularly, we asked:

Q_1. **What does one understand as a *better classification result*?** A better result should be one that increases a specific chosen metric. The problem, as discussed in § 5.1 (p. 47), is that the chosen dataset is highly unbalanced, with negative examples count overwhelmingly higher than positive ones. Metrics such as overall *Precision*, *Recall*, *Specificity* or *Sensitivity*, are not robust to label inbalance and thus a simple *Dummy Classifier* that, for example, always choosesdo *Not Malaria*, may easily reveal better than actually trained classifiers, under these metrics. As such, we opted to use the F_1 score, as discussed in § 5.9.2 (p. 57), and provided evidence that, in this case, it was significantly more robust.

Q_2. **How can one provide statistical evidence that one classifier is significantly better than another?** Due to the stochastic nature of training, and particularly due to the inherent

randomness of a cross-validation, simply comparing averages does not give one sufficient information to provide a definitive answer to the question **is A better than B, or did it happened by chance?** Hence, during this work, we opted to formulate this question as a classical *Hypothesis Test*, and resort to a *t-test* to determine if two sets of data — in this case, the scores of each validation fold – are **significantly** different from each other.

We now attempt to provide an answer to the author's fundamental research questions as stated in § 4.2 (p. 44):

H_1. Would the application of different *Feature Selection* methods, in order to reduce the *Dimensionality* of the dataset, lead us to better classification results?

H_2. Would the application of different *Classifiers* lead us to better classification results?

H_3. Would an optimization of the hyper-parameters of both the *Feature Selection* methods and used *Classifiers* lead us to better classification results?

The answer to all above questions is **yes**. As seen in § 5.1.3 (p. 49), and summarized in Table 5.1 (p. 59) both feature selection and different classifiers — and the corresponding tuning of the corresponding hyper-parameters — can lead to better results according to the evaluated F_1 score. In particular, selecting the most relevant features using **Variance Threshold** with $threshold = 0.10$, and subsequently training an **Ada Boost** ensemble classifier with $n\_estimators = 350$ leads to a *statistically significant* better result than previously obtained, as well as when compared to other combinations of selectors, classifiers, and hyper-parameters.

H_4. Would the final results be statistically and significantly better than the established baseline?

Although the answer is **yes**, as one can observe in both § 5.1.3 (p. 49) and Figure 5.6 (p. 60), the remaining question is *how much?* Indeed, one conclusion that can be drafted by looking at their final F_1 score, is that there is considerable evidence that further selection of features, classifiers, and hyper-parameters, would not lead to *much better* results. Notwithstanding, providing an answer to such question is not trivial — unless one reads the F_1 score too literaly — and was considered out-of-scope for this dissertation.

7.2 Main Contributions

Here we restate the goal drafted in § 4.2 (p. 44): ***"having as basis the dataset and results obtained by Rosado et al. [RCE+16], derive a systematic method that can (i) explore a wide space of feature selection methods, classifiers, and tuning of the hyperparameters, to provide a classifier able to identify the presence of malaria in microscopical images, and (ii) provide statistical evidence of the performance of each obtained classifier, when compared to others, by relying on robust metrics."*** To which, the result of this dissertation provides three main contributions:

1. **A systematic study** and a **statistical comparison** of several different machine learning methods that can be applied to detecting malaria parasites in microscopic images;
2. **A framework** that can be easily extended and used to support the analysis of future datasets one might have, and;
3. **An improved classifier** that can be easily trained, studied and used by resorting to the built framework.

7.3 Future Work

During the dissertation development, we have detected some shortcomings and identified several areas of improvement. Here's a list of the most important work to pursue in the future:

- **Explore the Grid Search module**, from the *Python*'s SCIKIT-LEARN package. *Grid Search* is a more known way of performing *hyperparameter* optimization, through an exhaustive searching of a manually specified subset of the *hyperparameter* space of a learning algorithm. Although this is not very different from what was done in this dissertation, this package already provides a tested and widely used mechanism that could easily be integrated in our workflow;
- **Explore the Probability Calibration module**, from the *Python*'s SCIKIT-LEARN package. When performing classification, one usually does not simply want to predict the class label, but also obtain a probability of the respective label in what concerns the classifier accuracy. Accessing this probability gives one confidence on the prediction. However, the probability one usually gets from training is optimized according to a threshold (*i.e.* above 0.5 is positive, and below is negative), and does not allow precise reasoning such as *10% of the cases the classifier gives a* $p = 0.1$ *of being positive, are, indeed, positive*. The calibration module in this package allows one to better calibrate the probabilities of a given model, or to add support for probability prediction. For models such as NAÏVE BAYES, which make unrealistic independence assumptions, it would be interesting to perform probability calibration and verify if the results are better;
- **Generate synthetic dataset entrances using *Sampling* techniques**. As described in Section 5.1 (p. 47), the used dataset is highly unbalanced. In these cases, it is common to make usage of *Oversampling* methods, *e.g. SMOTE*, *Random Over Sampling*, and *ADASYN*, to adjust the class distribution of a dataset, *i.e.* the ratio between the different classes/categories represented. Fortunately, there is also a *Python* package that includes not just the Over-, but several other sampling techniques[1];
- **Extend the framework to *multi-label classification***. As mentioned in Chapter Chapter 3 (p. 29), the first step of malaria parasites in blood smears is proceeded in thick films, where

[1] IMBALANCED-LEARN: http://contrib.scikit-learn.org/imbalanced-learn/index.html

one identifies if the patient has or not malaria. Then, the drop of blood from an infected sample is spread across a large area of the slide (thin blood smears), which helps one to discover what species of malaria is causing the infection. An approach to consider would be to extend the framework to this multi-label classification problem, having a pipeline that could give a full diagnosis on the disease, conducting the infected people to the right treatment;

- **Perform some feature engineering**. As stated in § 5.1.2 (p. 48), there is a great number of high correlated features in the dataset; one possibility could be to restructure the image processing module in order to extract other features that could provide more useful information to the classifier, using the work here developed to guide which ones are promising, and deleting those that have little to no impact;

- **Apply *Deep Learning* to this domain**. Recent works show that building a neural network that could be trained directly on images provides notably better results on classification. Applying this concept to our domain, with the recent and easy to use technologies that have emerged on this area (*cf.* § 2.7, p. 23), would be a next step to consider. It should be noted, however, that the dataset is *extremely* small when compared to the typical sizes used in *deep learning* approaches. One way to mitigate this would be to perform invariant-preserving transformations on the images, such as rotations, translations, and scaling, and thus synthesize expanded versions of the dataset.

7.4 Final Remarks

As someone who is doing Data Science for the first time — and learning machine learning on the go — I must say that the presented work was not easy for me... Pursuing this dissertation in an institution such as Fraunhofer, working on a real and big problem of our present time, it kind of felt a bit overwhelming sometimes. Still, in the end, it revealed to be everything I was looking for during the five years of my master degree: a combination of two areas that I love, in order to help people — specially the ill — and something to motivate my will and curiosity to keep up on the right track. In the meantime, Data Science opened my mind to *new paths for a new life* stage that *begins with the end* of this dissertation; I now have little doubts that my professional ambitions became focused on Scientific Research and to, very likely, enroll in a *Ph.D*.

References

[] Understanding the different types of variable in statistics. https://statistics.laerd.com/statistical-guides/types-of-variable.php (Cited on p. 49).

[AMM13] A. S. Abdul-Nasir, M. Y. Mashor, and Z. Mohamed. Colour image segmentation approach for detection of malaria parasites using various colour models and k-means clustering. In *WSEAS TRANSACTIONS on BIOLOGY and BIOMEDICINE*, volume 10, 2013 (Cited on p. 39).

[ANP+11] D. Anggraini, A. S. Nugroho, C. Pratama, I. E. Rozi, V. Pragesjvara, and M. Gunawan. Automated status identification of microscopic images obtained from malaria thin blood smears using bayes decision: a study case in plasmodium falciparum. In *2011 International Conference on Advanced Computer Science and Information Systems*, pages 347–352, 2011 (Cited on p. 39).

[Art15] C. Arteta. *Computer Vision and Machine Learning for Microscopy Image Analysis*. PhD thesis, University of Oxford, 2015 (Cited on p. 23).

[BC] Leo Breiman and Adele Cutler. Random forests - classification description. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm (Cited on pp. 9, 10).

[Bro] Jason Brownlee. Discover feature engineering. http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/ (Cited on p. 16).

[BY01] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 29:1165–1188, 2001 (Cited on p. 51).

[Che95] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995. DOI: 10.1109/34.400568 (Cited on p. 43).

[CN04] Rich Caruana and Alexandru Niculescu-Mizil. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In pages 69–78. ACM Press, 2004 (Cited on p. 57).

[CN06] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, Pittsburgh, Pennsylvania, USA. ACM, 2006. DOI: 10.1145/1143844.1143865 (Cited on p. 21).

[CS14] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28, 2014. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2013.11.024 (Cited on pp. 17, 18).

REFERENCES

[Dow11] Allen B. Downey. *Think Stats: Probability and Statistics for Programmers*. Green Tea Press, Needham, Massachusetts, 2011. URL: http://www.greenteapress.com/thinkstats/thinkstats.pdf (Cited on p. 58).

[EHZ11] M. Elter, E. Haßlmeyer, and T. Zerfaß. Detection of malaria parasites in thick blood films. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5140–5144, 2011 (Cited on pp. 23, 31).

[Exp] Explorable. Type i error and type ii error - experimental errors. https://explorable.com/type-i-error (Cited on p. 51).

[Fod02] Imola Fodor. A Survey of Dimension Reduction Techniques. Technical report, 2002 (Cited on p. 51).

[FXL+11] Y. Fang, W. Xiong, W. Lin, and Z. Chen. Unsupervised malaria parasite detection based on phase spectrum. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 7997–8000, 2011. DOI: 10.1109/IEMBS.2011.6091972 (Cited on p. 39).

[GDC+11] M. Ghosh, D. Das, C. Chakraborty, and A. K. Ray. Plasmodium vivax segmentation using modified fuzzy divergence. In *2011 International Conference on Image Information Processing*, pages 1–5, 2011. DOI: 10.1109/ICIIP.2011.6108873 (Cited on p. 40).

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1. URL: http://dx.doi.org/10.1007/s10994-006-6226-1 (Cited on p. 10).

[Gho] Debiprasad Ghosh. Ml branches. https://www.quora.com/What-is-the-difference-between-supervised-and-unsupervised-learning-algorithms (Cited on p. 16).

[GLF89] John H. Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. *Artif. Intell.*, 40(1-3):11–61, September 1989. ISSN: 00043702. DOI: 10.1016/0004-3702(89)90046-5. URL: http://dx.doi.org/10.1016/0004-3702(89)90046-5 (Cited on p. 48).

[HL05] Jin Huang and C. X. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310, 2005. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.50 (Cited on p. 57).

[IP] Statistics Introduction to Probability and Random Processes. Bivariate normal distribution | jointly normal. https://www.probabilitycourse.com/chapter5/5_3_2_bivariate_normal_dist.php (Cited on p. 49).

[JL95] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 1995 (Cited on pp. 10, 11).

[Ken12] Carolyn Masters Williams Kenrad E. Nelson. *Infectious Disease Epidemiology: Theory and Practice*. Jones & Bartlett Learning, 3rd edition, 2012. URL: https://libgen.unblocked.world/book/index.php?md5=93cfeee88b15b59cfdf2cca6699dc266 (Cited on p. 2).

[KIP+15] S. Kaewkamnerd, A. Intarapanich, M. Pannarat, S. Chaotheing, C. Uthaipibull, and S. Tongsima. Detection and classification device for malaria parasites in thick-blood films. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, volume 1, pages 435–438, 2015. DOI: 10.1109/IDAACS.2011.6072791 (Cited on pp. 23, 30, 31).

[KKM12] S. Kareem, I. Kale, and R. C. S. Morling. Automated p.falciparum detection system for post-treatment malaria diagnosis using modified annular ring ratio method. In *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, pages 432–436, 2012. DOI: 10.1109/UKSim.2012.108 (Cited on p. 39).

[KMK11] S. Kareem, R. C. S. Morling, and I. Kale. A novel method to count the red blood cells in thin blood films. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 1021–1024, 2011. DOI: 10.1109/ISCAS.2011.5937742 (Cited on pp. 38, 39).

[Kot07] S. B. Kotsiantis. Supervised machine learning: a review of classification techniques. *Informatica 31*, 2007:249–268, 2007 (Cited on pp. 9, 10, 12).

[KPL+14] N. A. Khan, H. Pervaz, A. K. Latif, A. Musharraf, and Saniya. Unsupervised identification of malaria parasites using computer vision. In *2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 263–267, 2014 (Cited on p. 39).

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. ISSN: 0028-0836. DOI: 10.1038/nature14539 (Cited on p. 21).

[MAB13] L. Malihi, K. Ansari-Asl, and A. Behbahani. Malaria parasite detection in giemsa-stained blood cell images. In *2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP)*, pages 360–365, 2013 (Cited on pp. 23, 35, 39).

[Mih] Christian Mihaescu. Naive bayes classification algorithm - faculty of automatics, computers and eletronics (university of craiova, romania). http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab4-NaiveBayes.pdf (Cited on p. 52).

[Min] Rapid Miner. Rapidminer platform - unified data science platform. https://rapidminer.com/products/ (Cited on p. 24).

[MKC+10] S. Mandal, A. Kumar, J. Chatterjee, M. Manjunatha, and A. K. Ray. Segmentation of blood smear images using normalized cuts for detection of malarial parasites. In *2010 Annual IEEE India Conference (INDICON)*, pages 1–4, 2010. DOI: 10.1109/INDCON.2010.5712739 (Cited on pp. 35, 36).

[Mon] Université de Montréal. Université de montréal - gpu research center. https://developer.nvidia.com/academia/centers/university-montreal (Cited on p. 26).

[MPM+05] F. Ellis McKenzie, Wendy A. Prudhomme, Alan J. Magill, J. Russ Forney, Barnyen Permpanich, Carmen Lucas, Robert A. Gasser, and Chansuda Wongsrichanalai. White blood cell counts and malaria. *J Infect Dis*, 192(2):323–330, July 2005. ISSN: 0022-1899. DOI: 10.1086/431152. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2481386/. 15962228[pmid] (Cited on p. 43).

[MR09] V. V. Makkapati and R. M. Rao. Segmentation of malaria parasites in peripheral blood smear images. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1361–1364, 2009 (Cited on p. 38).

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012 (Cited on pp. 5–7, 9, 15–19, 51, 53, 54).

[Org16] World Health Organization. World malaria report 2016. http://apps.who.int/iris/bitstream/10665/252038/1/9789241511711-eng.pdf, 2016 (Cited on p. 1).

[Ots79] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. DOI: 10.1109/TSMC.1979.4310076 (Cited on p. 42).

[Pro] R Project. What is r? https://www.r-project.org/about.html (Cited on p. 24).

[PRP13] I. K. E. Purnama, F. Z. Rahmanti, and M. H. Purnomo. Malaria parasite identification on thick blood film using genetic programming. In *2013 3rd International Conference on Instrumentation, Communications, Information Technology and Biomedical Engineering (ICICI-BME)*, pages 194–198, 2013. DOI: 10.1109/ICICI-BME.2013.6698491 (Cited on pp. 23, 32).

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2078195 (Cited on p. 24).

[Pyta] Python. Pandas-profiling 1.4.0. https://pypi.python.org/pypi/pandas-profiling (Cited on p. 48).

[Pytb] Scikit-Learn - Python. 1.13.1. removing features with low variance. http://scikit-learn.org/stable/modules/feature_selection.html (Cited on p. 51).

[Pytc] Scikit-Learn - Python. 3.3.6. dummy estimators. http://scikit-learn.org/stable/modules/model_evaluation.html#dummy-estimators (Cited on p. 52).

[QAM+14] John A. Quinn, Alfred Andama, Ian Munabi, and Fred N. Kiwanuka. Automated blood smear analysis for mobile malaria diagnosis. In *Mobile Point-of-Care Monitors and Diagnostic Device Design*. W. Karlen and K. Iniewski, CRC Press., 2014. DOI: http://dx.doi.org/10.1016/j.jbi.2008.11.005 (Cited on p. 35).

[Rasa] Sebastian Raschka. About feature scaling and normalization. http://sebastianraschka.com/Articles/2014_about_feature_scaling.html (Cited on p. 21).

[Rasb] Sebastian Raschka. Machine learning faq. https://sebastianraschka.com/faq/docs/feature_sele_categories.html (Cited on p. 18).

[RCE+16] Luis Rosado, Jose M. Correia da Costa, Dirk Elias, and Jaime S. Cardoso. Automated detection of malaria parasites on thick blood smears via mobile devices. *Procedia Computer Science*:138–144, 2016 (Cited on pp. 23, 41, 43–45, 48, 69, 70).

[ROV+17] Luis Rosado, João Oliveira, Maria João M. Vasconcelos, José M. Correia da Costa, Dirk Elias, and Jaime S. Cardoso. μsmartscope: 3d-printed smartphone microscope with motorized automated stage. In *Proceedings of the 10th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 1: BIODEVICES, (BIOSTEC 2017)*, pages 38–48. INSTICC, ScitePress, 2017. ISBN: 978-989-758-216-5. DOI: 10.5220/0006155800380048 (Cited on p. 42).

[SS] Christos Stergiou and Dimitrios Siganos. Neural networks. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.htm (Cited on p. 11).

[Sta] Laerd Statistics. Pearson's correlation. http://www.statstutor.ac.uk/resources/uploaded/pearsons.pdf (Cited on p. 49).

[Sys16] SAS - Statistical Analysis System. Machine learning - what it is & why it matters. http://www.sas.com/it_it/insights/analytics/machine-learning.html, 2016 (Cited on pp. 5, 6).

[TDK06] F. Boray Tek, Andrew G. Dempster, and Izzet Kale. Malaria parasite detection in peripheral blood images. In *in Proc. British Machine Vision Conference*, 2006 (Cited on p. 23).

[Ten] TensorFlow. Getting started with tensorflow. https://www.tensorflow.org/get_started/get_started (Cited on p. 25).

[The] Theano. Theano 0.8 release - welcome. http://deeplearning.net/software/theano/ (Cited on p. 26).

[TK07] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: an overview. *Int J Data Warehousing and Mining*, 2007:1–13, 2007 (Cited on p. 7).

[TKV08] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis P. Vlahavas. Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In *ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 2008. URL: http://mlkd.csd.auth.gr/publication%5C_details.asp?publicationID=276 (Cited on p. 7).

[Toa] Statistics How To. F statistic: definition and how to find it. https://www.statisticshowto.com/f-statistic (Cited on p. 51).

[Tob] Statistics How To. Linear relationship: definition, examples. http://www.statisticshowto.com/linear-relationship/ (Cited on p. 49).

[Unia] Standford University. Neural networks. http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/ (Cited on p. 12).

[Unib] Standford University. Pooling: overview. http://ufldl.stanford.edu/tutorial/supervised/Pooling/ (Cited on p. 13).

[Van16] Jake VanderPas. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, 2016 (Cited on p. 24).

[Wek] Weka. Weka 3: data mining software in java. http://weka.sourceforge.net (Cited on p. 25).

[Wika] Wikipedia. Autoencoder. https://en.wikipedia.org/wiki/Autoencoder (Cited on p. 19).

[Wikb] Wikipedia. Box plot. https://en.wikipedia.org/wiki/Box_plot (Cited on p. 55).

[Wikc] Wikipedia. Dimensionality reduction. https://en.wikipedia.org/wiki/Dimensionality_reduction (Cited on p. 16).

[Wikd] Wikipedia. Ensemble learning. https://en.wikipedia.org/wiki/Ensemble_learning (Cited on pp. 15, 52).

[Wike] Wikipedia. False discovery rate. http://https://en.wikipedia.org/wiki/False_discovery_rate (Cited on p. 51).

[Wikf] Wikipedia. Feature scaling. https://en.wikipedia.org/wiki/Feature_scaling (Cited on p. 21).

[Wikg] Wikipedia. Feature selection. https://en.wikipedia.org/wiki/Feature_selection (Cited on pp. 17, 18).

[Wikh] Wikipedia. Hyperparameter optimization. https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning) (Cited on p. 53).

[Wiki] Wikipedia. K-nearest neighbors algorithm. http://https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (Cited on p. 52).

[Wikj] Wikipedia. Multilayer perceptron. http://https://en.wikipedia.org/wiki/Multilayer_perceptron (Cited on p. 52).

[Wikk] Wikipedia. Perceptron. http://https://en.wikipedia.org/wiki/Perceptron (Cited on p. 52).

[Wikl] Wikipedia. Random forest. https://en.wikipedia.org/wiki/Random_forest (Cited on p. 9).

[Wikm] Wikipedia. Statistical classification. https://en.wikipedia.org/wiki/Statistical_classification (Cited on p. 7).

[Wikn] Wikipedia. Supervised learning. https://en.wikipedia.org/wiki/Supervised_learning (Cited on p. 6).

[Wiko] Wikipedia. Support vector machine. http://https://en.wikipedia.org/wiki/Support_vector_machine (Cited on p. 52).

[WZA06] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, 2006. ISSN: 0146-4833. DOI: 10.1145/1163593.1163596 (Cited on p. 22).

[YAM12] Leonardo Yunda, Andrés Alarcón, and Jorge Millán. Automated image analysis method for p-vivax malaria parasite detection in thick film blood images. *Sistemas & Telemática*, 10(20):9–25, 2012. DOI: 10.18046/syt.v10i20.1151 (Cited on pp. 33, 34).

[ZCZ+10] L. h. Zou, J. Chen, J. Zhang, and N. Garcia. Malaria cell counting diagnosis within large field of view. In *2010 International Conference on Digital Image Computing: Techniques and Applications*, pages 172–177, 2010. DOI: 10.1109/DICTA.2010.40 (Cited on pp. 37, 38).

[Zhe15] Alice Zheng. *Evaluate Machine Learning Models - A Beginner's Guide to Key Concepts and Pitfalls*. O'Reilly Media, Inc, 1st edition, 2015 (Cited on p. 15).

Appendix A

Preliminary Dataset Analysis

Profile report

23/06/2017, 14:29

Overview

Dataset info

Number of variables	317
Number of observations	2653
Total Missing (%)	0.0%
Total size in memory	6.4 MiB
Average record size in memory	2.5 KiB

Variables types

Numeric	167
Categorical	0
Date	0
Text (Unique)	0
Rejected	150

Warnings

ChromatineGLCM_D0_Entropy is highly correlated with Chromatine_GLRLM_D135GLNU (ρ = 0.92124) Rejected
ChromatineGLCM_D135_Contrast is highly correlated with ChromatineGLCM_D45_Contrast (ρ = 0.98021) Rejected
ChromatineGLCM_D135_Correl is highly correlated with ChromatineGLCM_D45_Correl (ρ = 0.96286) Rejected
ChromatineGLCM_D135_Energy is highly correlated with ChromatineGLCM_D90_Energy (ρ = 0.93377) Rejected
ChromatineGLCM_D135_Entropy is highly correlated with ChromatineGLCM_D90_Entropy (ρ = 0.97837) Rejected
ChromatineGLCM_D45_Contrast is highly correlated with ChromatineGLCM_D0_Contrast (ρ = 0.98248) Rejected
ChromatineGLCM_D45_Correl is highly correlated with ChromatineGLCM_D0_Correl (ρ = 0.94556) Rejected
ChromatineGLCM_D45_Energy is highly correlated with ChromatineGLCM_D0_Energy (ρ = 0.95624) Rejected
ChromatineGLCM_D45_Entropy is highly correlated with ChromatineGLCM_D0_Entropy (ρ = 0.98524) Rejected
ChromatineGLCM_D90_Energy is highly correlated with ChromatineGLCM_D45_Energy (ρ = 0.93689) Rejected
ChromatineGLCM_D90_Entropy is highly correlated with ChromatineGLCM_D45_Entropy (ρ = 0.97921) Rejected
Chromatine_DFTMIN has 51 / 1.9% zeros
Chromatine_Entropy_Lab_Chroma is highly correlated with Chromatine_L2Norm_Lab_Hue (ρ = 0.91096) Rejected
Chromatine_Entropy_Lab_Hue is highly correlated with Chromatine_blobEquivalentDiameter (ρ = 0.91197) Rejected
Chromatine_GLRLM_D0GLNU is highly correlated with Chromatine_blobBoundingBoxArea (ρ = 0.93752) Rejected
Chromatine_GLRLM_D0LRHGRE is highly correlated with Chromatine_GLRLM_D0LRE (ρ = 0.92101) Rejected
Chromatine_GLRLM_D0LRLGLE is highly correlated with Chromatine_GLRLM_D0LRE (ρ = 0.99539) Rejected
Chromatine_GLRLM_D0RP is highly correlated with Chromatine_blobMinDiameter (ρ = 0.91804) Rejected
Chromatine_GLRLM_D0SRHGE is highly correlated with Chromatine_GLRLM_D0SRLGE (ρ = 0.96111) Rejected
Chromatine_GLRLM_D0SRLGE is highly correlated with Chromatine_GLRLM_D0SRE (ρ = 0.99822) Rejected
Chromatine_GLRLM_D135GLNU is highly correlated with Chromatine_GLRLM_D90LRHGRE (ρ = 0.98184) Rejected
Chromatine_GLRLM_D135LRE is highly correlated with Chromatine_GLRLM_D90LRHGRE (ρ = 0.93082) Rejected
Chromatine_GLRLM_D135LRHGRE is highly correlated with Chromatine_GLRLM_D135LRLGLE (ρ = 0.94081) Rejected
Chromatine_GLRLM_D135LRLGLE is highly correlated with Chromatine_GLRLM_D135LRE (ρ = 0.99722) Rejected
Chromatine_GLRLM_D135RLNU is highly correlated with Chromatine_GLRLM_D90RLNU (ρ = 0.96329) Rejected
Chromatine_GLRLM_D135RP is highly correlated with Chromatine_GLRLM_D45RP (ρ = 0.98344) Rejected
Chromatine_GLRLM_D135SRE is highly correlated with Chromatine_GLRLM_D90SRHGE (ρ = 0.98087) Rejected
Chromatine_GLRLM_D135SRHGE is highly correlated with Chromatine_GLRLM_D135SRLGE (ρ = 0.98546) Rejected
Chromatine_GLRLM_D135SRLGE is highly correlated with Chromatine_GLRLM_D135SRE (ρ = 0.99944) Rejected
Chromatine_GLRLM_D45GLNU is highly correlated with Chromatine_GLRLM_D0GLNU (ρ = 0.98171) Rejected
Chromatine_GLRLM_D45LRE is highly correlated with Chromatine_GLRLM_D0LRHGRE (ρ = 0.92211) Rejected
Chromatine_GLRLM_D45LRHGRE is highly correlated with Chromatine_GLRLM_D45LRLGLE (ρ = 0.93905) Rejected
Chromatine_GLRLM_D45LRLGLE is highly correlated with Chromatine_GLRLM_D45LRE (ρ = 0.99735) Rejected
Chromatine_GLRLM_D45RLNU is highly correlated with Chromatine_GLRLM_D0RLNU (ρ = 0.96043) Rejected
Chromatine_GLRLM_D45SRE is highly correlated with Chromatine_GLRLM_D0SRHGE (ρ = 0.97891) Rejected
Chromatine_GLRLM_D45SRHGE is highly correlated with Chromatine_GLRLM_D45SRLGE (ρ = 0.97881) Rejected
Chromatine_GLRLM_D45SRLGE is highly correlated with Chromatine_GLRLM_D45SRE (ρ = 0.99903) Rejected
Chromatine_GLRLM_D90GLNU is highly correlated with Chromatine_GLRLM_D45GLNU (ρ = 0.98081) Rejected
Chromatine_GLRLM_D90LRE is highly correlated with Chromatine_GLRLM_D45LRHGRE (ρ = 0.9607) Rejected
Chromatine_GLRLM_D90LRHGRE is highly correlated with Chromatine_GLRLM_D90LRE (ρ = 0.92981) Rejected
Chromatine_GLRLM_D90LRLGLE is highly correlated with Chromatine_GLRLM_D90LRE (ρ = 0.99511) Rejected
Chromatine_GLRLM_D90RLNU is highly correlated with Chromatine_GLRLM_D45RLNU (ρ = 0.96051) Rejected
Chromatine_GLRLM_D90RP is highly correlated with Chromatine_blobMaxDiameter (ρ = 0.93131) Rejected
Chromatine_GLRLM_D90SRE is highly correlated with Chromatine_GLRLM_D45SRHGE (ρ = 0.98862) Rejected
Chromatine_GLRLM_D90SRHGE is highly correlated with Chromatine_GLRLM_D90SRLGE (ρ = 0.96381) Rejected
Chromatine_GLRLM_D90SRLGE is highly correlated with Chromatine_GLRLM_D90SRE (ρ = 0.99835) Rejected
Chromatine_HorizGrayEntropy_Lab_Chroma is highly correlated with Chromatine_HorizGrayEnergy_Lab_Chroma (ρ = 0.91651) Rejected

file:///Users/mafaldafalcaotvf/Documents/Dev/GitHub/Dissertation/FrameworkDev/descriptiveDatasetAnalysis.html

Page 1 of 51

Chromatine_HorizGrayEntropy_Lab_Hue is highly skewed (γ1 = 25.379)
Chromatine_HorizGrayKurtosis_Lab_Chroma is highly correlated with Chromatine_HorizGraySkewness_Lab_Chroma (ρ = 0.96019) Rejected
Chromatine_HorizGrayKurtosis_Lab_Hue is highly skewed (γ1 = 33.308)
Chromatine_HorizGrayL2Norm_Lab_Chroma is highly correlated with Chromatine_HorizGrayL1Norm_Lab_Chroma (ρ = 0.97554) Rejected
Chromatine_HorizGrayL2Norm_Lab_Hue is highly correlated with Chromatine_HorizGrayL1Norm_Lab_Hue (ρ = 0.96549) Rejected
Chromatine_HorizGraySTD_Lab_Chroma is highly correlated with Chromatine_HorizGrayMean_Lab_Chroma (ρ = 0.95134) Rejected
Chromatine_HorizGraySTD_Lab_Hue is highly correlated with Chromatine_HorizGrayMean_Lab_Hue (ρ = 0.95194) Rejected
Chromatine_HorizGraySkewness_Lab_Chroma has 35 / 1.3% zeros
Chromatine_HorizGraySkewness_Lab_Chroma is highly skewed (γ1 = 38.038)
Chromatine_HorizGraySkewness_Lab_Hue has 35 / 1.3% zeros
Chromatine_HorizGraySkewness_Lab_Hue is highly skewed (γ1 = 45.409)
Chromatine_Kurtosis_Lab_Chroma is highly skewed (γ1 = 45.239)
Chromatine_Kurtosis_Lab_Hue is highly skewed (γ1 = 44.514)
Chromatine_L1Norm_Lab_Chroma is highly correlated with Chromatine_Entropy_Lab_Chroma (ρ = 0.99739) Rejected
Chromatine_L1Norm_Lab_Hue is highly correlated with Chromatine_Entropy_Lab_Hue (ρ = 0.97778) Rejected
Chromatine_L2Norm_Lab_Chroma is highly correlated with Chromatine_L1Norm_Lab_Chroma (ρ = 0.98845) Rejected
Chromatine_L2Norm_Lab_Hue is highly correlated with Chromatine_L1Norm_Lab_Hue (ρ = 0.97437) Rejected
Chromatine_LaplacianMAX is highly correlated with Chromatine_LaplacianSTD (ρ = 0.94578) Rejected
Chromatine_LaplacianSTD is highly correlated with Chromatine_LaplacianMEAN (ρ = 0.94747) Rejected
Chromatine_Mean_Lab_Chroma is highly correlated with Chromatine_Energy_Lab_Chroma (ρ = 0.99934) Rejected
Chromatine_Mean_Lab_Hue is highly correlated with Chromatine_Energy_Lab_Hue (ρ = 0.99565) Rejected
Chromatine_Skewness_Lab_Chroma is highly skewed (γ1 = -33.403)
Chromatine_Skewness_Lab_Hue is highly skewed (γ1 = -39.136)
Chromatine_VertGrayEntropy_Lab_Chroma is highly correlated with Chromatine_VertGrayEnergy_Lab_Chroma (ρ = 0.93093) Rejected
Chromatine_VertGrayKurtosis_Lab_Chroma is highly correlated with Chromatine_Kurtosis_Lab_Chroma (ρ = 0.97525) Rejected
Chromatine_VertGrayKurtosis_Lab_Hue is highly skewed (γ1 = 39.398)
Chromatine_VertGrayL2Norm_Lab_Chroma is highly correlated with Chromatine_VertGrayL1Norm_Lab_Chroma (ρ = 0.97758) Rejected
Chromatine_VertGrayL2Norm_Lab_Hue is highly correlated with Chromatine_VertGrayL1Norm_Lab_Hue (ρ = 0.96695) Rejected
Chromatine_VertGraySTD_Lab_Chroma is highly correlated with Chromatine_VertGrayMean_Lab_Chroma (ρ = 0.95972) Rejected
Chromatine_VertGraySTD_Lab_Hue is highly correlated with Chromatine_VertGrayMean_Lab_Hue (ρ = 0.94754) Rejected
Chromatine_blobAsymmetryIndex has 237 / 8.9% zeros
Chromatine_blobBoundingBoxArea is highly correlated with Chromatine_blobConvexHullArea (ρ = 0.94424) Rejected
Chromatine_blobConvexHullArea is highly correlated with Chromatine_blobPerimeter (ρ = 0.97383) Rejected
Chromatine_blobEllipticalSymmetry has 791 / 29.8% zeros
Chromatine_blobElongation is highly correlated with Chromatine_blobEccentricity (ρ = 0.90214) Rejected
Chromatine_blobEquivalentDiameter is highly correlated with Chromatine_blobBoundingBoxArea (ρ = 0.90872) Rejected
Chromatine_blobIrregularityIndexB is highly correlated with Chromatine_blobPrincipalAxisRatio (ρ = 0.97484) Rejected
Chromatine_blobIrregularityIndexC is highly correlated with Chromatine_blobBoundingBoxRatio (ρ = 0.95231) Rejected
Chromatine_blobIrregularityIndexD is highly correlated with Chromatine_blobIrregularityIndexC (ρ = 0.94749) Rejected
Chromatine_blobMajorAsymmetryIndex has 521 / 19.6% zeros
Chromatine_blobMinorAsymmetryIndex has 525 / 19.8% zeros
Chromatine_blobPerimeter is highly correlated with Chromatine_blobArea (ρ = 0.95579) Rejected
CytoplasmGLCM_D135_Contrast is highly correlated with CytoplasmGLCM_D90_Contrast (ρ = 0.91269) Rejected
CytoplasmGLCM_D135_Correl is highly correlated with CytoplasmGLCM_D45_Correl (ρ = 0.99327) Rejected
CytoplasmGLCM_D135_Energy is highly correlated with CytoplasmGLCM_D45_Energy (ρ = 0.99009) Rejected
CytoplasmGLCM_D135_Entropy is highly correlated with CytoplasmGLCM_D45_Entropy (ρ = 0.9955) Rejected
CytoplasmGLCM_D135_Homogeneity is highly correlated with CytoplasmGLCM_D45_Homogeneity (ρ = 0.93183) Rejected
CytoplasmGLCM_D45_Contrast is highly correlated with CytoplasmGLCM_D0_Contrast (ρ = 0.99365) Rejected
CytoplasmGLCM_D45_Correl is highly correlated with CytoplasmGLCM_D0_Correl (ρ = 0.98976) Rejected
CytoplasmGLCM_D45_Energy is highly correlated with CytoplasmGLCM_D0_Energy (ρ = 0.90412) Rejected
CytoplasmGLCM_D45_Entropy is highly correlated with CytoplasmGLCM_D0_Entropy (ρ = 0.94157) Rejected
CytoplasmGLCM_D45_Homogeneity is highly correlated with CytoplasmGLCM_D0_Homogeneity (ρ = 0.92184) Rejected
CytoplasmGLCM_D90_Contrast is highly correlated with CytoplasmGLCM_D45_Contrast (ρ = 0.91471) Rejected
Cytoplasm_DFTMAX is highly correlated with Cytoplasm_blobBoundingBoxArea (ρ = 0.90089) Rejected
Cytoplasm_Entropy_Lab_Chroma is highly correlated with Cytoplasm_L1Norm_Lab_Hue (ρ = 0.95441) Rejected
Cytoplasm_GLRLM_D0GLNU is highly correlated with Cytoplasm_DFTMAX (ρ = 0.904) Rejected
Cytoplasm_GLRLM_D0LRLGE is highly correlated with Cytoplasm_GLRLM_D0LRE (ρ = 0.98151) Rejected
Cytoplasm_GLRLM_D0RP is highly correlated with Cytoplasm_blobMinDiameter (ρ = 0.95676) Rejected
Cytoplasm_GLRLM_D0SRHGE is highly correlated with Cytoplasm_GLRLM_D0SRE (ρ = 0.92618) Rejected
Cytoplasm_GLRLM_D0SRLGE is highly correlated with Cytoplasm_GLRLM_D0SRE (ρ = 0.99723) Rejected
Cytoplasm_GLRLM_D135GLNU is highly correlated with Cytoplasm_GLRLM_D90GLNU (ρ = 0.98711) Rejected
Cytoplasm_GLRLM_D135LRE is highly correlated with Cytoplasm_GLRLM_D90LRLGE (ρ = 0.98024) Rejected

Cytoplasm_GLRLM_D135LRHGRE is highly correlated with Cytoplasm_GLRLM_D90LRHGRE (ρ = 0.93819) Rejected
Cytoplasm_GLRLM_D135LRLGE is highly correlated with Cytoplasm_GLRLM_D135LRE (ρ = 0.98926) Rejected
Cytoplasm_GLRLM_D135RLNU is highly correlated with Cytoplasm_GLRLM_D90RLNU (ρ = 0.99754) Rejected
Cytoplasm_GLRLM_D135RP is highly correlated with Cytoplasm_GLRLM_D45RP (ρ = 0.99635) Rejected
Cytoplasm_GLRLM_D135SRE is highly correlated with Cytoplasm_GLRLM_D90SRHGE (ρ = 0.93615) Rejected
Cytoplasm_GLRLM_D135SRHGE is highly correlated with Cytoplasm_GLRLM_D135SRLGE (ρ = 0.95598) Rejected
Cytoplasm_GLRLM_D135SRLGE is highly correlated with Cytoplasm_GLRLM_D135SRE (ρ = 0.9987) Rejected
Cytoplasm_GLRLM_D45GLNU is highly correlated with Cytoplasm_GLRLM_D0GLNU (ρ = 0.98691) Rejected
Cytoplasm_GLRLM_D45LRE is highly correlated with Cytoplasm_GLRLM_D0LRLGE (ρ = 0.98042) Rejected
Cytoplasm_GLRLM_D45LRHGRE is highly correlated with Cytoplasm_GLRLM_D0LRHGRE (ρ = 0.93607) Rejected
Cytoplasm_GLRLM_D45LRLGE is highly correlated with Cytoplasm_GLRLM_D45LRE (ρ = 0.98939) Rejected
Cytoplasm_GLRLM_D45RLNU is highly correlated with Cytoplasm_GLRLM_D0RLNU (ρ = 0.99782) Rejected
Cytoplasm_GLRLM_D45SRE is highly correlated with Cytoplasm_GLRLM_D0SRHGE (ρ = 0.92896) Rejected
Cytoplasm_GLRLM_D45SRHGE is highly correlated with Cytoplasm_GLRLM_D45SRLGE (ρ = 0.95578) Rejected
Cytoplasm_GLRLM_D45SRLGE is highly correlated with Cytoplasm_GLRLM_D45SRE (ρ = 0.99858) Rejected
Cytoplasm_GLRLM_D90GLNU is highly correlated with Cytoplasm_GLRLM_D45GLNU (ρ = 0.98684) Rejected
Cytoplasm_GLRLM_D90LRE is highly correlated with Cytoplasm_GLRLM_D45LRLGE (ρ = 0.98889) Rejected
Cytoplasm_GLRLM_D90LRHGRE is highly correlated with Cytoplasm_GLRLM_D45LRHGRE (ρ = 0.93769) Rejected
Cytoplasm_GLRLM_D90LRLGE is highly correlated with Cytoplasm_GLRLM_D90LRE (ρ = 0.98136) Rejected
Cytoplasm_GLRLM_D90RLNU is highly correlated with Cytoplasm_GLRLM_D45RLNU (ρ = 0.99777) Rejected
Cytoplasm_GLRLM_D90RP is highly correlated with Cytoplasm_blobMaxDiameter (ρ = 0.95381) Rejected
Cytoplasm_GLRLM_D90SRE is highly correlated with Cytoplasm_GLRLM_D45SRHGE (ρ = 0.97095) Rejected
Cytoplasm_GLRLM_D90SRHGE is highly correlated with Cytoplasm_GLRLM_D90SRLGE (ρ = 0.90716) Rejected
Cytoplasm_GLRLM_D90SRLGE is highly correlated with Cytoplasm_GLRLM_D90SRE (ρ = 0.9974) Rejected
Cytoplasm_HorizGrayEntropy_Lab_Chroma is highly correlated with Cytoplasm_HorizGrayEnergy_Lab_Chroma (ρ = 0.91664) Rejected
Cytoplasm_HorizGrayKurtosis_Lab_Chroma is highly skewed (γ1 = 29.241)
Cytoplasm_HorizGrayL2Norm_Lab_Chroma is highly correlated with Cytoplasm_HorizGrayL1Norm_Lab_Chroma (ρ = 0.96587) Rejected
Cytoplasm_HorizGrayL2Norm_Lab_Hue is highly correlated with Cytoplasm_HorizGrayL1Norm_Lab_Hue (ρ = 0.96773) Rejected
Cytoplasm_HorizGrayMean_Lab_Hue is highly correlated with Cytoplasm_HorizGrayEnergy_Lab_Hue (ρ = 0.92699) Rejected
Cytoplasm_HorizGraySTD_Lab_Chroma is highly correlated with Cytoplasm_HorizGrayMean_Lab_Chroma (ρ = 0.95511) Rejected
Cytoplasm_HorizGraySTD_Lab_Hue is highly correlated with Cytoplasm_HorizGrayMean_Lab_Hue (ρ = 0.93832) Rejected
Cytoplasm_Kurtosis_Lab_Chroma is highly skewed (γ1 = 33.346)
Cytoplasm_Kurtosis_Lab_Hue is highly skewed (γ1 = 34.023)
Cytoplasm_L1Norm_Lab_Chroma is highly correlated with Cytoplasm_Entropy_Lab_Chroma (ρ = 0.99762) Rejected
Cytoplasm_L1Norm_Lab_Hue is highly correlated with Cytoplasm_Entropy_Lab_Hue (ρ = 0.96361) Rejected
Cytoplasm_L2Norm_Lab_Chroma is highly correlated with Cytoplasm_L1Norm_Lab_Chroma (ρ = 0.99419) Rejected
Cytoplasm_L2Norm_Lab_Hue is highly correlated with Cytoplasm_L1Norm_Lab_Hue (ρ = 0.97197) Rejected
Cytoplasm_LaplacianSTD is highly correlated with Cytoplasm_LaplacianMEAN (ρ = 0.94073) Rejected
Cytoplasm_Mean_Lab_Chroma is highly correlated with Cytoplasm_Energy_Lab_Chroma (ρ = 0.99959) Rejected
Cytoplasm_Mean_Lab_Hue is highly correlated with Cytoplasm_Energy_Lab_Hue (ρ = 0.99716) Rejected
Cytoplasm_Skewness_Lab_Chroma is highly skewed (γ1 = 27.865)
Cytoplasm_VertGrayEntropy_Lab_Chroma is highly correlated with Cytoplasm_VertGrayEnergy_Lab_Chroma (ρ = 0.92486) Rejected
Cytoplasm_VertGrayKurtosis_Lab_Chroma is highly skewed (γ1 = 47.488)
Cytoplasm_VertGrayKurtosis_Lab_Hue is highly skewed (γ1 = 26.211)
Cytoplasm_VertGrayL2Norm_Lab_Chroma is highly correlated with Cytoplasm_VertGrayL1Norm_Lab_Chroma (ρ = 0.96511) Rejected
Cytoplasm_VertGrayL2Norm_Lab_Hue is highly correlated with Cytoplasm_VertGrayL1Norm_Lab_Hue (ρ = 0.96905) Rejected
Cytoplasm_VertGrayMean_Lab_Hue is highly correlated with Cytoplasm_VertGrayEnergy_Lab_Hue (ρ = 0.92395) Rejected
Cytoplasm_VertGraySTD_Lab_Chroma is highly correlated with Cytoplasm_VertGrayMean_Lab_Chroma (ρ = 0.94961) Rejected
Cytoplasm_VertGraySTD_Lab_Hue is highly correlated with Cytoplasm_VertGrayMean_Lab_Hue (ρ = 0.9359) Rejected
Cytoplasm_blobAsymmetryIndex is highly correlated with Cytoplasm_blobMinorAsymmetryIndex (ρ = 0.98728) Rejected
Cytoplasm_blobAsymmetry_Celebi2 is highly correlated with Cytoplasm_blobAsymmetry_Celebi1 (ρ = 0.92159) Rejected
Cytoplasm_blobBoundingBoxArea is highly correlated with Cytoplasm_blobConvexHullArea (ρ = 0.90039) Rejected
Cytoplasm_blobCircularity is highly correlated with Cytoplasm_blobSolidity (ρ = 0.90825) Rejected
Cytoplasm_blobConvexHullArea is highly correlated with Cytoplasm_blobPerimeter (ρ = 0.94681) Rejected
Cytoplasm_blobEquivalentDiameter is highly correlated with Cytoplasm_blobConvexHullArea (ρ = 0.90651) Rejected
Cytoplasm_blobIrregularityIndexB is highly correlated with Cytoplasm_blobPrincipalAxisRatio (ρ = 0.97148) Rejected
Cytoplasm_blobIrregularityIndexC is highly correlated with Cytoplasm_blobBoundingBoxRatio (ρ = 0.93237) Rejected
Cytoplasm_blobIrregularityIndexD is highly correlated with Cytoplasm_blobIrregularityIndexC (ρ = 0.95221) Rejected
Cytoplasm_blobMinorAsymmetryIndex is highly correlated with Cytoplasm_blobMajorAsymmetryIndex (ρ = 0.94962) Rejected
Ratio_LAB_Chroma_Mean2 is highly skewed (γ1 = 41.355)
Ratio_LAB_Hue_Mean2 is highly skewed (γ1 = 25.776)
Ratio_blobBoundingBoxArea is highly correlated with Ratio_blobConvexHullArea (ρ = 0.94282) Rejected
Ratio_blobConvexHullArea is highly correlated with Ratio_blobPerimeter (ρ = 0.95667) Rejected

`Ratio_blobPerimeter` is highly correlated with `Ratio_blobArea` (ρ = 0.90072) Rejected
`ID` is highly correlated with `index` (ρ = 1) Rejected

Variables

ChromatineGLCM_D0_Cor
Numeric

Distinct count	2627	**Mean**	593.5
Unique (%)	99.0%	**Minimum**	10.867
Missing (%)	0.0%	**Maximum**	9585.1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D0_Con
Numeric

Distinct count	2652	**Mean**	0.25845
Unique (%)	100.0%	**Minimum**	-0.65048
Missing (%)	0.0%	**Maximum**	0.95973
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D0_Ene
Numeric

Distinct count	466	**Mean**	0.014397
Unique (%)	17.6%	**Minimum**	0.0026125
Missing (%)	0.0%	**Maximum**	0.044444
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~ChromatineGLCM_D0_Entro~~
Highly correlated

This variable is highly correlated with `Chromatine_GLRLM_D135GLNU` and should be ignored for analysis

Correlation 0.92124

ChromatineGLCM_D0_Hon
Numeric

Distinct count	2649	**Mean**	0.056489
Unique (%)	99.8%	**Minimum**	0.00020293
Missing (%)	0.0%	**Maximum**	0.53793
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D0_Max
Numeric

Distinct count	85	**Mean**	0.024646
Unique (%)	3.2%	**Minimum**	0.0046296

Missing (%)	0.0%	**Maximum**	0.11111
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~ChromatineGLCM_D135_Cor~~
Highly correlated

This variable is highly correlated with `ChromatineGLCM_D45_Contrast` and should be ignored for analysis

Correlation 0.98921

~~ChromatineGLCM_D135_Cor~~
Highly correlated

This variable is highly correlated with `ChromatineGLCM_D45_Correl` and should be ignored for analysis

Correlation 0.96286

~~ChromatineGLCM_D135_Ene~~
Highly correlated

This variable is highly correlated with `ChromatineGLCM_D90_Energy` and should be ignored for analysis

Correlation 0.93377

~~ChromatineGLCM_D135_Ent~~
Highly correlated

This variable is highly correlated with `ChromatineGLCM_D90_Entropy` and should be ignored for analysis

Correlation 0.97837

ChromatineGLCM_D135_H
Numeric

Distinct count	2649	**Mean**	0.053314
Unique (%)	99.8%	**Minimum**	0.0001729
Missing (%)	0.0%	**Maximum**	0.32187
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D135_M
Numeric

Distinct count	76	**Mean**	0.025493
Unique (%)	2.9%	**Minimum**	0.0050505
Missing (%)	0.0%	**Maximum**	0.13333
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D45_Cont Highly correlated	*This variable is highly correlated with ChromatineGLCM_D0_Contrast and should be ignored for analysis*	**Correlation**	0.98248
ChromatineGLCM_D45_Corr Highly correlated	*This variable is highly correlated with ChromatineGLCM_D0_Correl and should be ignored for analysis*	**Correlation**	0.94556
ChromatineGLCM_D45_Ener Highly correlated	*This variable is highly correlated with ChromatineGLCM_D0_Energy and should be ignored for analysis*	**Correlation**	0.95624
ChromatineGLCM_D45_Entr Highly correlated	*This variable is highly correlated with ChromatineGLCM_D0_Entropy and should be ignored for analysis*	**Correlation**	0.98524

ChromatineGLCM_D45_Ho
Numeric

Distinct count	2647	**Mean**	0.05383
Unique (%)	99.8%	**Minimum**	0.00023204
Missing (%)	0.0%	**Maximum**	0.32075
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D45_Ma
Numeric

Distinct count	76	**Mean**	0.025742
Unique (%)	2.9%	**Minimum**	0.0050505
Missing (%)	0.0%	**Maximum**	0.13333
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D90_Co
Numeric

Distinct count	2625	**Mean**	585.37
Unique (%)	98.9%	**Minimum**	5.5667
Missing (%)	0.0%	**Maximum**	12501
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D90_Co
Numeric

Distinct count	2653	**Mean**	0.23133
Unique (%)	100.0%	**Minimum**	-0.65811
Missing (%)	0.0%	**Maximum**	0.9845
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D90_Ene Highly correlated	*This variable is highly correlated with ChromatineGLCM_D45_Energy and should be ignored for analysis*	**Correlation**	0.93689
ChromatineGLCM_D90_Entr Highly correlated	*This variable is highly correlated with ChromatineGLCM_D45_Entropy and should be ignored for analysis*	**Correlation**	0.97921

ChromatineGLCM_D90_Ho
Numeric

Distinct count	2649	**Mean**	0.054958
Unique (%)	99.8%	**Minimum**	0.00021865
Missing (%)	0.0%	**Maximum**	0.64464
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

ChromatineGLCM_D90_Ma
Numeric

Distinct count	82	**Mean**	0.024407
Unique (%)	3.1%	**Minimum**	0.0045455
Missing (%)	0.0%	**Maximum**	0.10714
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_DFTMAX
Numeric

Distinct count	2298	**Mean**	8.74
Unique (%)	86.6%	**Minimum**	7.3658
Missing (%)	0.0%	**Maximum**	10.878
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_DFTMEAN
Numeric

Distinct count	2626	**Mean**	3.8983
Unique (%)	99.0%	**Minimum**	2.6931
Missing (%)	0.0%	**Maximum**	5.7402
Missing (n)	0	**Zeros (%)**	0.0%

Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_DFTMIN
Numeric

Distinct count	1070	**Mean**	1.3662
Unique (%)	40.3%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	4.0197
Missing (n)	0	**Zeros (%)**	1.9%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_DFTSTD
Numeric

Distinct count	2572	**Mean**	1.247
Unique (%)	96.9%	**Minimum**	0.83493
Missing (%)	0.0%	**Maximum**	1.795
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Energy_Lab_C
Numeric

Distinct count	2594	**Mean**	32929
Unique (%)	97.8%	**Minimum**	24922
Missing (%)	0.0%	**Maximum**	45063
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Energy_Lab_H
Numeric

Distinct count	2603	**Mean**	1591.2
Unique (%)	98.1%	**Minimum**	389.02
Missing (%)	0.0%	**Maximum**	2672.4
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Entropy_Lab_
Highly correlated

This variable is highly correlated with `Chromatine_L2Norm_Lab_Hue` and should be ignored for analysis

Correlation 0.91096

Chromatine_Entropy_Lab_
Highly correlated

This variable is highly correlated with `Chromatine_blobEquivalentDiameter` and should be ignored for analysis

Correlation 0.91197

Chromatine_GLRLM_D0GLN
Highly correlated

This variable is highly correlated with `Chromatine_blobBoundingBoxArea` and should be ignored for analysis

Correlation 0.93752

Chromatine_GLRLM_D0HO
Numeric

Distinct count	154	**Mean**	1.1097
Unique (%)	5.8%	**Minimum**	1
Missing (%)	0.0%	**Maximum**	3
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D0LG
Numeric

Distinct count	157	**Mean**	0.97509
Unique (%)	5.9%	**Minimum**	0.74523
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D0LR
Numeric

Distinct count	2628	**Mean**	16065
Unique (%)	99.1%	**Minimum**	1651.9
Missing (%)	0.0%	**Maximum**	39966
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D0LRH
Highly correlated

This variable is highly correlated with `Chromatine_GLRLM_D0LRE` and should be ignored for analysis

Correlation 0.92101

Chromatine_GLRLM_D0LRL
Highly correlated

This variable is highly correlated with `Chromatine_GLRLM_D0LRE` and should be ignored for analysis

Correlation 0.99539

Chromatine_GLRLM_D0RL
Numeric

Distinct count	841	**Mean**	1.8489
Unique (%)	31.7%	**Minimum**	1
Missing (%)	0.0%	**Maximum**	8.1163
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_GLRLM_D0RP~~ Highly correlated	*This variable is highly correlated with Chromatine_blobMinDiameter and should be ignored for analysis*	**Correlation**	0.91604

Chromatine_GLRLM_D0SR
Numeric

Distinct count	2641	**Mean**	8.0375e-05
Unique (%)	99.5%	**Minimum**	2.5266e-05
Missing (%)	0.0%	**Maximum**	0.00093341
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_GLRLM_D0SRHG~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D0SRLGE and should be ignored for analysis*	**Correlation**	0.9611
~~Chromatine_GLRLM_D0SRLG~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D0SRE and should be ignored for analysis*	**Correlation**	0.99822
~~Chromatine_GLRLM_D135GL~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D90GLNU and should be ignored for analysis*	**Correlation**	0.98184

Chromatine_GLRLM_D135
Numeric

Distinct count	91	**Mean**	1.0641
Unique (%)	3.4%	**Minimum**	1
Missing (%)	0.0%	**Maximum**	1.6
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D135
Numeric

Distinct count	91	**Mean**	0.98451
Unique (%)	3.4%	**Minimum**	0.85
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		

		Infinite (n)	0

Toggle details

~~Chromatine_GLRLM_D135LR~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D90LRHGRE and should be ignored for analysis*	**Correlation**	0.93082
~~Chromatine_GLRLM_D135LR~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D135LRLGE and should be ignored for analysis*	**Correlation**	0.94081
~~Chromatine_GLRLM_D135LR~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D135LRE and should be ignored for analysis*	**Correlation**	0.99722
~~Chromatine_GLRLM_D135RL~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D90RLNU and should be ignored for analysis*	**Correlation**	0.96329
~~Chromatine_GLRLM_D135RP~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D45RP and should be ignored for analysis*	**Correlation**	0.99344
~~Chromatine_GLRLM_D135SR~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D90SRHGE and should be ignored for analysis*	**Correlation**	0.98087
~~Chromatine_GLRLM_D135SR~~ Highly correlated	*This variable is highly correlated with Chromatine_GLRLM_D135SRLGE and should be ignored for analysis*	**Correlation**	0.98546

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D135SRE and should be ignored for analysis — Correlation 0.9994

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D0GLNU and should be ignored for analysis — Correlation 0.98171

Chromatine_GLRLM_D45H
Numeric

Distinct count	95	Mean	1.0643
Unique (%)	3.6%	Minimum	1
Missing (%)	0.0%	Maximum	1.7
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D45L
Numeric

Distinct count	96	Mean	0.98452
Unique (%)	3.6%	Minimum	0.87037
Missing (%)	0.0%	Maximum	1
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D0LRHGRE and should be ignored for analysis — Correlation 0.92211

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D45LRLGE and should be ignored for analysis — Correlation 0.93905

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D45LRE and should be ignored for analysis — Correlation 0.99735

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D0RLNU and should be ignored for analysis — Correlation 0.96043

Chromatine_GLRLM_D45R
Numeric

Distinct count	96	Mean	0.031374
Unique (%)	3.6%	Minimum	0.019531
Missing (%)	0.0%	Maximum	0.084375
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D0SRHGE and should be ignored for analysis — Correlation 0.97891

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D45SRLGE and should be ignored for analysis — Correlation 0.97881

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D45SRE and should be ignored for analysis — Correlation 0.99903

Highly correlated — This variable is highly correlated with Chromatine_GLRLM_D45GLNU and should be ignored for analysis — Correlation 0.98081

Chromatine_GLRLM_D90H
Numeric

Distinct count	162	Mean	1.1085
Unique (%)	6.1%	Minimum	1
Missing (%)	0.0%	Maximum	1.946
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D90L
Numeric

Distinct count	166	**Mean**	0.97513
Unique (%)	6.3%	**Minimum**	0.79354
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_GLRLM_D90LRE
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D45LRHGRE and should be ignored for analysis

Correlation 0.9607

Chromatine_GLRLM_D90LRH
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D90LRE and should be ignored for analysis

Correlation 0.92981

Chromatine_GLRLM_D90LRL
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D90LRE and should be ignored for analysis

Correlation 0.99511

Chromatine_GLRLM_D90RLN
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D45RLNU and should be ignored for analysis

Correlation 0.9605

Chromatine_GLRLM_D90RP
Highly correlated

This variable is highly correlated with Chromatine_blobMaxDiameter and should be ignored for analysis

Correlation 0.93131

Chromatine_GLRLM_D90SRE
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D45SRHGE and should be ignored for analysis

Correlation 0.98862

Chromatine_GLRLM_D90SRH
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D90SRLGE

Correlation 0.96381

and should be ignored for analysis

Chromatine_GLRLM_D90SRL
Highly correlated

This variable is highly correlated with Chromatine_GLRLM_D90SRE and should be ignored for analysis

Correlation 0.99835

Chromatine_HorizGrayEne
Numeric

Distinct count	2625	**Mean**	34.078
Unique (%)	98.9%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	1642.8
Missing (n)	0	**Zeros (%)**	0.9%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_HorizGrayEne
Numeric

Distinct count	2627	**Mean**	12.513
Unique (%)	99.0%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	942.37
Missing (n)	0	**Zeros (%)**	0.8%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_HorizGrayEn
Highly correlated

This variable is highly correlated with Chromatine_HorizGrayEnergy_Lab_Chroma and should be ignored for analysis

Correlation 0.91651

Chromatine_HorizGrayEntr
Numeric

Distinct count	2628	**Mean**	1002.4
Unique (%)	99.1%	**Minimum**	-4.3242
Missing (%)	0.0%	**Maximum**	248560
Missing (n)	0	**Zeros (%)**	0.8%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_HorizGrayKu
Highly correlated

This variable is highly correlated with Chromatine_HorizGraySkewness_Lab_Chroma and should be ignored for analysis

Correlation 0.96019

Chromatine_HorizGrayKurt
Numeric

Distinct count	2626	**Mean**	1791000
Unique (%)	99.0%	**Minimum**	0

Missing (%)	0.0%	Maximum	1932100000	
Missing (n)	0	Zeros (%)	0.9%	
Infinite (%)	0.0%			
Infinite (n)	0			

Toggle details

Chromatine_HorizGrayL1N
Numeric

Distinct count	2615	Mean	50.424
Unique (%)	98.6%	Minimum	0
Missing (%)	0.0%	Maximum	1322.4
Missing (n)	0	Zeros (%)	0.9%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_HorizGrayL1N
Numeric

Distinct count	2622	Mean	30.208
Unique (%)	98.8%	Minimum	0
Missing (%)	0.0%	Maximum	1046.2
Missing (n)	0	Zeros (%)	0.8%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_HorizGrayL2N~~
Highly correlated

This variable is highly correlated with Chromatine_HorizGrayL1Norm_Lab_Chroma and should be ignored for analysis

Correlation 0.97554

~~Chromatine_HorizGrayL2N~~
Highly correlated

This variable is highly correlated with Chromatine_HorizGrayL1Norm_Lab_Hue and should be ignored for analysis

Correlation 0.96549

Chromatine_HorizGrayMea
Numeric

Distinct count	2622	Mean	3.269
Unique (%)	98.8%	Minimum	0
Missing (%)	0.0%	Maximum	36.647
Missing (n)	0	Zeros (%)	0.9%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_HorizGrayMea
Numeric

Distinct count	2624	Mean	2.101
Unique (%)	98.9%	Minimum	0
Missing (%)	0.0%	Maximum	28.277
Missing (n)	0	Zeros (%)	0.8%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_HorizGraySTI~~
Highly correlated

This variable is highly correlated with Chromatine_HorizGrayMean_Lab_Chroma and should be ignored for analysis

Correlation 0.95134

~~Chromatine_HorizGraySTI~~
Highly correlated

This variable is highly correlated with Chromatine_HorizGrayMean_Lab_Hue and should be ignored for analysis

Correlation 0.95194

Chromatine_HorizGraySke
Numeric

Distinct count	2617	Mean	58693
Unique (%)	98.6%	Minimum	-8088700
Missing (%)	0.0%	Maximum	81215000
Missing (n)	0	Zeros (%)	1.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_HorizGraySke
Numeric

Distinct count	2615	Mean	2028.4
Unique (%)	98.6%	Minimum	-53109
Missing (%)	0.0%	Maximum	2905600
Missing (n)	0	Zeros (%)	1.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Kurtosis_Lab_
Numeric

Distinct count	2643	Mean	5622400
Unique (%)	99.6%	Minimum	0
Missing (%)	0.0%	Maximum	8090600000
Missing (n)	0	Zeros (%)	0.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Kurtosis_Lab_
Numeric

Distinct count	2644	Mean	54347
Unique (%)	99.7%	Minimum	0
Missing (%)	0.0%	Maximum	86428000
Missing (n)	0	Zeros (%)	0.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_L1Norm_Lab_C~~
Highly correlated

This variable is highly correlated with Chromatine_Entropy_Lab_Chroma and should be ignored for analysis **Correlation** 0.99739

~~Chromatine_L1Norm_Lab_H~~
Highly correlated
This variable is highly correlated with Chromatine_Entropy_Lab_Hue and should be ignored for analysis **Correlation** 0.97778

~~Chromatine_L2Norm_Lab_C~~
Highly correlated
This variable is highly correlated with Chromatine_L1Norm_Lab_Chroma and should be ignored for analysis **Correlation** 0.98845

~~Chromatine_L2Norm_Lab_H~~
Highly correlated
This variable is highly correlated with Chromatine_L1Norm_Lab_Hue and should be ignored for analysis **Correlation** 0.97437

~~Chromatine_LaplacianMAX~~
Highly correlated
This variable is highly correlated with Chromatine_LaplacianSTD and should be ignored for analysis **Correlation** 0.94578

Chromatine_LaplacianMEA
Numeric

Distinct count	1218	**Mean**	5.6871
Unique (%)	45.9%	**Minimum**	1.3125
Missing (%)	0.0%	**Maximum**	21.422
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_LaplacianSTD~~
Highly correlated
This variable is highly correlated with Chromatine_LaplacianMEAN and should be ignored for analysis **Correlation** 0.94747

~~Chromatine_Mean_Lab_Chr~~
Highly correlated
This variable is highly correlated with Chromatine_Energy_Lab_Chroma **Correlation** 0.99934

and should be ignored for analysis

~~Chromatine_Mean_Lab_Hue~~
Highly correlated
This variable is highly correlated with Chromatine_Energy_Lab_Hue and should be ignored for analysis **Correlation** 0.99565

Chromatine_STD_Lab_Chr
Numeric

Distinct count	2640	**Mean**	1.5468
Unique (%)	99.5%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	15.998
Missing (n)	0	**Zeros (%)**	0.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_STD_Lab_Hue
Numeric

Distinct count	2640	**Mean**	1.0076
Unique (%)	99.5%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	9.3647
Missing (n)	0	**Zeros (%)**	0.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Skewness_Lal
Numeric

Distinct count	2643	**Mean**	-1255
Unique (%)	99.6%	**Minimum**	-9853600
Missing (%)	0.0%	**Maximum**	3529500
Missing (n)	0	**Zeros (%)**	0.4%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_Skewness_Lal
Numeric

Distinct count	2642	**Mean**	-66.748
Unique (%)	99.6%	**Minimum**	-120610
Missing (%)	0.0%	**Maximum**	6411.4
Missing (n)	0	**Zeros (%)**	0.3%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_VertGrayEnerg
Numeric

Distinct count	2638	**Mean**	41.682
Unique (%)	99.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	3141.5
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_VertGrayEnerg
Numeric

Distinct count	2637	**Mean**	13.63
Unique (%)	99.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	651.32
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_VertGrayEntr~~
Highly correlated

This variable is highly correlated with Chromatine_VertGrayEnergy_Lab_Chroma and should be ignored for analysis

Correlation 0.93093

Chromatine_VertGrayEntro
Numeric

Distinct count	2638	**Mean**	992.49
Unique (%)	99.4%	**Minimum**	-4.6578
Missing (%)	0.0%	**Maximum**	146710
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_VertGrayKurt~~
Highly correlated

This variable is highly correlated with Chromatine_Kurtosis_Lab_Chroma and should be ignored for analysis

Correlation 0.97525

Chromatine_VertGrayKurto
Numeric

Distinct count	2638	**Mean**	443730
Unique (%)	99.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	436890000
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

1e8

Toggle details

Chromatine_VertGrayL1No
Numeric

Distinct count	2636	**Mean**	56.469
Unique (%)	99.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	1512.9
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_VertGrayL1No
Numeric

Distinct count	2636	**Mean**	31.141
Unique (%)	99.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	759.51

Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_VertGrayL2Nc~~
Highly correlated

This variable is highly correlated with Chromatine_VertGrayL1Norm_Lab_Chroma and should be ignored for analysis

Correlation 0.97758

~~Chromatine_VertGrayL2Nc~~
Highly correlated

This variable is highly correlated with Chromatine_VertGrayL1Norm_Lab_Hue and should be ignored for analysis

Correlation 0.96695

Chromatine_VertGrayMean
Numeric

Distinct count	2635	**Mean**	3.6607
Unique (%)	99.3%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	50.431
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_VertGrayMean
Numeric

Distinct count	2637	**Mean**	2.2408
Unique (%)	99.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	23.015
Missing (n)	0	**Zeros (%)**	0.5%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_VertGraySTD_~~
Highly correlated

This variable is highly correlated with Chromatine_VertGrayMean_Lab_Chroma and should be ignored for analysis

Correlation 0.95972

~~Chromatine_VertGraySTD_~~
Highly correlated

This variable is highly correlated with Chromatine_VertGrayMean_Lab_Hue and should be ignored for analysis

Correlation 0.94754

Chromatine_VertGraySkew
Numeric

Distinct count	2625	**Mean**	1537.5
Unique (%)	98.9%	**Minimum**	-10133000
Missing (%)	0.0%	**Maximum**	8363000
Missing (n)	0	Zeros (%)	1.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_VertGraySkew
Numeric

Distinct count	2627	**Mean**	743.81
Unique (%)	99.0%	**Minimum**	-220040
Missing (%)	0.0%	**Maximum**	304590
Missing (n)	0	Zeros (%)	0.9%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobArea
Numeric

Distinct count	927	**Mean**	1.0757
Unique (%)	34.9%	**Minimum**	0.40779
Missing (%)	0.0%	**Maximum**	3.4836
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobAsymmeti
Numeric

Distinct count	313	**Mean**	0.2482
Unique (%)	11.8%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	0.84615
Missing (n)	0	**Zeros (%)**	8.9%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobAsymmeti
Numeric

Distinct count	178	**Mean**	0.64907
Unique (%)	6.7%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	Zeros (%)	0.2%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobAsymmeti
Numeric

Distinct count	328	**Mean**	1.5178
Unique (%)	12.4%	**Minimum**	0.475
Missing (%)	0.0%	**Maximum**	2
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobBounding
Highly correlated

This variable is highly correlated with Chromatine_blobConvexHullArea and should be ignored for analysis

Correlation 0.94424

Chromatine_blobBounding
Numeric

Distinct count	72	**Mean**	1.1
Unique (%)	2.7%	**Minimum**	0.2
Missing (%)	0.0%	**Maximum**	5.5
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobCircularity
Numeric

Distinct count	461	**Mean**	1.4039
Unique (%)	17.4%	**Minimum**	0.32271
Missing (%)	0.0%	**Maximum**	1.9291
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobCompacti
Numeric

Distinct count	462	**Mean**	0.77238
Unique (%)	17.4%	**Minimum**	0.51838
Missing (%)	0.0%	**Maximum**	3.0988
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobConvexHu
Highly correlated

This variable is highly correlated with Chromatine_blobPerimeter and should be ignored for analysis

Correlation 0.97383

Chromatine_blobEccentric
Numeric

Distinct count	1052	**Mean**	2.8038
Unique (%)	39.7%	**Minimum**	1.0101
Missing (%)	0.0%	**Maximum**	39.763
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobEllipticalS
Numeric

Distinct count	303	**Mean**	0.23885
Unique (%)	11.4%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	29.8%
Infinite (%)	0.0%		

Infinite (n) 0

Toggle details

~~Chromatine_blobElongati~~
Highly correlated

This variable is highly correlated with Chromatine_blobEccentricity and should be ignored for analysis

Correlation 0.90214

~~Chromatine_blobEquivale~~
Highly correlated

This variable is highly correlated with Chromatine_blobBoundingBoxArea and should be ignored for analysis

Correlation 0.90872

Chromatine_blobExtent
Numeric

Distinct count	251	**Mean**	1.1811
Unique (%)	9.5%	**Minimum**	0.26389
Missing (%)	0.0%	**Maximum**	2.2
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobIrregularit
Numeric

Distinct count	453	**Mean**	0.72633
Unique (%)	17.1%	**Minimum**	0.43454
Missing (%)	0.0%	**Maximum**	1.0879
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_blobIrregula~~
Highly correlated

This variable is highly correlated with Chromatine_blobPrincipalAxisRatio and should be ignored for analysis

Correlation 0.97484

~~Chromatine_blobIrregula~~
Highly correlated

This variable is highly correlated with Chromatine_blobBoundingBoxRatio and should be ignored for analysis

Correlation 0.95231

~~Chromatine_blobIrregula~~
Highly correlated

This variable is highly correlated with Chromatine_blobIrregularityIndexC

Correlation 0.94749

and should be ignored for analysis

Chromatine_blobLengthen
Numeric

Distinct count	1056	**Mean**	0.49704
Unique (%)	39.8%	**Minimum**	0.025149
Missing (%)	0.0%	**Maximum**	0.98999
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobMajorAsy
Numeric

Distinct count	232	**Mean**	0.24995
Unique (%)	8.7%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	19.6%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobMajorAsy
Numeric

Distinct count	173	**Mean**	0.75491
Unique (%)	6.5%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	Zeros (%)	0.2%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobMaxDiam
Numeric

Distinct count	362	**Mean**	0.22794
Unique (%)	13.6%	**Minimum**	0.051733
Missing (%)	0.0%	**Maximum**	0.98395
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobMinDiame
Numeric

Distinct count	357	**Mean**	0.22473
Unique (%)	13.5%	**Minimum**	0.092166
Missing (%)	0.0%	**Maximum**	0.92308
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobMinorAsy
Numeric

Distinct count	230	**Mean**	0.24644
Unique (%)	8.7%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	0.91304
Missing (n)	0	**Zeros (%)**	19.8%
Infinite (%)	0.0%		

	Infinite (n)	0			

Toggle details

Chromatine_blobMinorAsy
Numeric

Distinct count	149	**Mean**	0.76294
Unique (%)	5.6%	**Minimum**	0.045455
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Chromatine_blobPerimete~~
Highly correlated

This variable is highly correlated with `Chromatine_blobArea` and should be ignored for analysis

Correlation 0.95579

Chromatine_blobPrincipalA
Numeric

Distinct count	72	**Mean**	1.0809
Unique (%)	2.7%	**Minimum**	0.18182
Missing (%)	0.0%	**Maximum**	5
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobRadialVar
Numeric

Distinct count	2482	**Mean**	19.97
Unique (%)	93.6%	**Minimum**	0.23865
Missing (%)	0.0%	**Maximum**	357.86
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Chromatine_blobSolidity
Numeric

Distinct count	278	**Mean**	1.1095
Unique (%)	10.5%	**Minimum**	0.6701
Missing (%)	0.0%	**Maximum**	1.8
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D0_Cont
Numeric

Distinct count	2648	**Mean**	325.84
Unique (%)	99.8%	**Minimum**	5.2729
Missing (%)	0.0%	**Maximum**	10048
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D0_Corr
Numeric

Distinct count	2651	**Mean**	0.45397
Unique (%)	99.9%	**Minimum**	-0.51695
Missing (%)	0.0%	**Maximum**	0.96982
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D0_Ener
Numeric

Distinct count	2568	**Mean**	0.0033883
Unique (%)	96.8%	**Minimum**	0.00051443
Missing (%)	0.0%	**Maximum**	0.013785
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D0_Entr
Numeric

Distinct count	2623	**Mean**	8.6766
Unique (%)	98.9%	**Minimum**	6.8714
Missing (%)	0.0%	**Maximum**	11.159
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D0_Hom
Numeric

Distinct count	2647	**Mean**	0.094024
Unique (%)	99.8%	**Minimum**	0.00020715
Missing (%)	0.0%	**Maximum**	0.52661
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D0_Max
Numeric

Distinct count	923	**Mean**	0.01115
Unique (%)	34.8%	**Minimum**	0.0021044
Missing (%)	0.0%	**Maximum**	0.062963
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~CytoplasmGLCM_D135_Con~~
Highly correlated

This variable is highly correlated with `CytoplasmGLCM_D90_Contrast` and should be ignored for analysis

Correlation 0.91269

~~CytoplasmGLCM_D135_Cor~~
Highly correlated

This variable is highly correlated with `CytoplasmGLCM_D45_Correl`

Correlation 0.99327

and should be ignored for analysis

~~CytoplasmGLCM_D135_Ener~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D45_Energy and should be ignored for analysis

Correlation 0.99009

~~CytoplasmGLCM_D135_Entr~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D45_Entropy and should be ignored for analysis

Correlation 0.9955

~~CytoplasmGLCM_D135_Homo~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D45_Homogeneity and should be ignored for analysis

Correlation 0.93183

CytoplasmGLCM_D135_Ma
Numeric

Distinct count	873	**Mean**	0.0089998
Unique (%)	32.9%	**Minimum**	0.0014148
Missing (%)	0.0%	**Maximum**	0.041152
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~CytoplasmGLCM_D45_Contr~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D0_Contrast and should be ignored for analysis

Correlation 0.99365

~~CytoplasmGLCM_D45_Corre~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D0_Correl and should be ignored for analysis

Correlation 0.98976

~~CytoplasmGLCM_D45_Energ~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D0_Energy and should be ignored for analysis

Correlation 0.90412

~~CytoplasmGLCM_D45_Entro~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D0_Entropy and should be ignored for analysis

Correlation 0.94157

~~CytoplasmGLCM_D45_Homog~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D0_Homogeneity and should be ignored for analysis

Correlation 0.92184

CytoplasmGLCM_D45_Ma
Numeric

Distinct count	877	**Mean**	0.0089964
Unique (%)	33.1%	**Minimum**	0.0015444
Missing (%)	0.0%	**Maximum**	0.040293
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~CytoplasmGLCM_D90_Contr~~
Highly correlated

This variable is highly correlated with CytoplasmGLCM_D45_Contrast and should be ignored for analysis

Correlation 0.91471

CytoplasmGLCM_D90_Cor
Numeric

Distinct count	2646	**Mean**	0.44208
Unique (%)	99.7%	**Minimum**	-0.50306
Missing (%)	0.0%	**Maximum**	0.96615
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D90_Ene
Numeric

Distinct count	2563	**Mean**	0.0033594
Unique (%)	96.6%	**Minimum**	0.00059508
Missing (%)	0.0%	**Maximum**	0.017148
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D90_Ent
Numeric

Distinct count	2625	**Mean**	8.6887
Unique (%)	98.9%	**Minimum**	6.9211
Missing (%)	0.0%	**Maximum**	10.833
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		

Infinite (n)	0		

Toggle details

CytoplasmGLCM_D90_Hor
Numeric

Distinct count	2651	**Mean**	0.092055
Unique (%)	99.9%	**Minimum**	0.0002625
Missing (%)	0.0%	**Maximum**	0.4188
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

CytoplasmGLCM_D90_Ma:
Numeric

Distinct count	953	**Mean**	0.011154
Unique (%)	35.9%	**Minimum**	0.0017391
Missing (%)	0.0%	**Maximum**	0.063636
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_DFTMAX~~
Highly correlated

This variable is highly correlated with Cytoplasm_blobBoundingBoxArea and should be ignored for analysis

Correlation 0.90089

Cytoplasm_DFTMEAN
Numeric

Distinct count	2631	**Mean**	4.3604
Unique (%)	99.2%	**Minimum**	3.1086
Missing (%)	0.0%	**Maximum**	5.929
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_DFTMIN
Numeric

Distinct count	2359	**Mean**	1.0791
Unique (%)	88.9%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	4.6548
Missing (n)	0	Zeros (%)	0.6%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_DFTSTD
Numeric

Distinct count	2593	**Mean**	1.2406
Unique (%)	97.7%	**Minimum**	0.91237
Missing (%)	0.0%	**Maximum**	1.8384
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_Energy_Lab_Cl
Numeric

Distinct count	2582	**Mean**	33022
Unique (%)	97.3%	**Minimum**	28586
Missing (%)	0.0%	**Maximum**	41377
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_Energy_Lab_Hi
Numeric

Distinct count	2612	**Mean**	1635.9
Unique (%)	98.5%	**Minimum**	476.65
Missing (%)	0.0%	**Maximum**	2377.6
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_Entropy_Lab_(~~
Highly correlated

This variable is highly correlated with Cytoplasm_L1Norm_Lab_Hue and should be ignored for analysis

Correlation 0.95441

Cytoplasm_Entropy_Lab_H
Numeric

Distinct count	2640	**Mean**	2181700
Unique (%)	99.5%	**Minimum**	381740
Missing (%)	0.0%	**Maximum**	6391400
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_GLRLM_D0GLNU~~
Highly correlated

This variable is highly correlated with Cytoplasm_DFTMAX and should be ignored for analysis

Correlation 0.904

Cytoplasm_GLRLM_D0HG
Numeric

Distinct count	2220	**Mean**	1.3996
Unique (%)	83.7%	**Minimum**	1.027
Missing (%)	0.0%	**Maximum**	2.4439
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D0LGI
Numeric

Distinct count	2346	**Mean**	0.92782
Unique (%)	88.4%	**Minimum**	0.80906
Missing (%)	0.0%	**Maximum**	0.99324
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		

Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D0LRE
Numeric

Distinct count	2627	**Mean**	20609
Unique (%)	99.0%	**Minimum**	2869.9
Missing (%)	0.0%	**Maximum**	43896
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D0LRH
Numeric

Distinct count	2640	**Mean**	29683
Unique (%)	99.5%	**Minimum**	3461.9
Missing (%)	0.0%	**Maximum**	86311
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D0LRLGE
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D0LRE and should be ignored for analysis

Correlation 0.98151

Cytoplasm_GLRLM_D0RLN
Numeric

Distinct count	2628	**Mean**	12.681
Unique (%)	99.1%	**Minimum**	2.0714
Missing (%)	0.0%	**Maximum**	124.84
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D0RP
Highly correlated

This variable is highly correlated with Cytoplasm_blobMinDiameter and should be ignored for analysis

Correlation 0.95676

Cytoplasm_GLRLM_D0SRE
Numeric

Distinct count	2642	**Mean**	5.656e-05
Unique (%)	99.6%	**Minimum**	2.2893e-05
Missing (%)	0.0%	**Maximum**	0.00050481
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D0SRHGE
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D0SRE and should be ignored for analysis

Correlation 0.92618

Cytoplasm_GLRLM_D0SRLGE
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D0SRE and should be ignored for analysis

Correlation 0.99723

Cytoplasm_GLRLM_D135GLN
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D90GLNU and should be ignored for analysis

Correlation 0.98711

Cytoplasm_GLRLM_D135H
Numeric

Distinct count	1921	**Mean**	1.245
Unique (%)	72.4%	**Minimum**	1
Missing (%)	0.0%	**Maximum**	1.9857
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D135L
Numeric

Distinct count	1987	**Mean**	0.94897
Unique (%)	74.9%	**Minimum**	0.84137
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D135LRE
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D90LRLGE and should be ignored for analysis

Correlation 0.98024

Cytoplasm_GLRLM_D135LRH
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D90LRHGRE and should be ignored for analysis

Correlation 0.93819

Cytoplasm_GLRLM_D135LRL
Highly correlated

This variable is highly correlated with Cytoplasm_GLRLM_D135LRE

Correlation 0.98926

and should be ignored for analysis

Cytoplasm_GLRLM_D135RLN Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D90RLNU and should be ignored for analysis*	**Correlation**	0.99754
Cytoplasm_GLRLM_D135RP Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D45RP and should be ignored for analysis*	**Correlation**	0.99635
Cytoplasm_GLRLM_D135SRE Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D90SRHGE and should be ignored for analysis*	**Correlation**	0.93615
Cytoplasm_GLRLM_D135SRL Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D135SRLGE and should be ignored for analysis*	**Correlation**	0.95598
Cytoplasm_GLRLM_D135SRL Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D135SRE and should be ignored for analysis*	**Correlation**	0.9987
Cytoplasm_GLRLM_D45GLNU Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D0GLNU and should be ignored for analysis*	**Correlation**	0.98691

Cytoplasm_GLRLM_D45HG
Numeric

Distinct count	1965	**Mean**	1.2461
Unique (%)	74.1%	**Minimum**	1
Missing (%)	0.0%	**Maximum**	1.9708
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D45LG
Numeric

Distinct count	2040	**Mean**	0.94907
Unique (%)	76.9%	**Minimum**	0.84433
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D45LRE Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D0LRLGE and should be ignored for analysis*	**Correlation**	0.98042
Cytoplasm_GLRLM_D45LRHG Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D0LRHGRE and should be ignored for analysis*	**Correlation**	0.93607
Cytoplasm_GLRLM_D45LRLG Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D45LRE and should be ignored for analysis*	**Correlation**	0.98939
Cytoplasm_GLRLM_D45RLNU Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D0RLNU and should be ignored for analysis*	**Correlation**	0.99782

Cytoplasm_GLRLM_D45RP
Numeric

Distinct count	1333	**Mean**	0.094549
Unique (%)	50.2%	**Minimum**	0.045247
Missing (%)	0.0%	**Maximum**	0.27734
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D45SRE Highly correlated	*This variable is highly correlated with Cytoplasm_GLRLM_D0SRHGE and should be ignored for analysis*	**Correlation**	0.92896

~~Cytoplasm_GLRLM_D45SRHG~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45SRLGE and should be ignored for analysis
Correlation 0.95578

~~Cytoplasm_GLRLM_D45SRLG~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45SRE and should be ignored for analysis
Correlation 0.99858

~~Cytoplasm_GLRLM_D90GLNU~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45GLNU and should be ignored for analysis
Correlation 0.98684

Cytoplasm_GLRLM_D90HG
Numeric

Distinct count	2214	**Mean**	1.3964
Unique (%)	83.5%	**Minimum**	1.0216
Missing (%)	0.0%	**Maximum**	2.6868
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_GLRLM_D90LG
Numeric

Distinct count	2353	**Mean**	0.92751
Unique (%)	88.7%	**Minimum**	0.76591
Missing (%)	0.0%	**Maximum**	0.9946
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_GLRLM_D90LRE~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45LRLGE and should be ignored for analysis
Correlation 0.98889

~~Cytoplasm_GLRLM_D90LRHG~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45LRHGRE and should be ignored for analysis
Correlation 0.93769

~~Cytoplasm_GLRLM_D90LRLG~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D90LRE and should be ignored for analysis
Correlation 0.98136

~~Cytoplasm_GLRLM_D90RLNU~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45RLNU and should be ignored for analysis
Correlation 0.99777

~~Cytoplasm_GLRLM_D90RP~~
Highly correlated
This variable is highly correlated with Cytoplasm_blobMaxDiameter and should be ignored for analysis
Correlation 0.95381

~~Cytoplasm_GLRLM_D90SRE~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D45SRHGE and should be ignored for analysis
Correlation 0.97095

~~Cytoplasm_GLRLM_D90SRHG~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D90SRLGE and should be ignored for analysis
Correlation 0.90716

~~Cytoplasm_GLRLM_D90SRLG~~
Highly correlated
This variable is highly correlated with Cytoplasm_GLRLM_D90SRE and should be ignored for analysis
Correlation 0.9974

Cytoplasm_HorizGrayEnerg
Numeric

Distinct count	2650	**Mean**	75.847
Unique (%)	99.9%	**Minimum**	0.10785
Missing (%)	0.0%	**Maximum**	3687.9
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_HorizGrayEnerg
Numeric

Distinct count	2651	**Mean**	24.649
Unique (%)	99.9%	**Minimum**	0.060651
Missing (%)	0.0%	**Maximum**	573.61

	Missing (n)	0	**Zeros (%)**	0.0%
	Infinite (%)	0.0%		
	Infinite (n)	0		

Toggle details

~~Cytoplasm_HorizGrayEntr~~
Highly correlated

This variable is highly correlated with ~~Cytoplasm_HorizGrayEnergy_Lab_Chroma~~ and should be ignored for analysis

Correlation 0.91664

Cytoplasm_HorizGrayEntro
Numeric

Distinct count	2653	**Mean**	18443
Unique (%)	100.0%	**Minimum**	-31.715
Missing (%)	0.0%	**Maximum**	537890
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_HorizGrayKurto
Numeric

Distinct count	2652	**Mean**	1289000000
Unique (%)	100.0%	**Minimum**	0.0017472
Missing (%)	0.0%	**Maximum**	883420000000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

1e12

Toggle details

Cytoplasm_HorizGrayKurto
Numeric

Distinct count	2653	**Mean**	2529300
Unique (%)	100.0%	**Minimum**	1.7456e-06
Missing (%)	0.0%	**Maximum**	790730000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

1e9

Toggle details

Cytoplasm_HorizGrayL1Nc
Numeric

Distinct count	2648	**Mean**	682.72
Unique (%)	99.8%	**Minimum**	8.0593
Missing (%)	0.0%	**Maximum**	8740
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_HorizGrayL1Nc
Numeric

Distinct count	2649	**Mean**	477.45
Unique (%)	99.8%	**Minimum**	7.612
Missing (%)	0.0%	**Maximum**	4684.6
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		

	Infinite (n)	0	

Toggle details

~~Cytoplasm_HorizGrayL2Nc~~
Highly correlated

This variable is highly correlated with ~~Cytoplasm_HorizGrayL1Norm_Lab_Chroma~~ and should be ignored for analysis

Correlation 0.96587

~~Cytoplasm_HorizGrayL2Nc~~
Highly correlated

This variable is highly correlated with ~~Cytoplasm_HorizGrayL1Norm_Lab_Hue~~ and should be ignored for analysis

Correlation 0.96773

Cytoplasm_HorizGrayMear
Numeric

Distinct count	2648	**Mean**	5.1352
Unique (%)	99.8%	**Minimum**	0.083951
Missing (%)	0.0%	**Maximum**	53.567
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_HorizGrayMear~~
Highly correlated

This variable is highly correlated with ~~Cytoplasm_HorizGrayEnergy_Lab_Hue~~ and should be ignored for analysis

Correlation 0.92699

~~Cytoplasm_HorizGraySTD_~~
Highly correlated

This variable is highly correlated with ~~Cytoplasm_HorizGrayMean_Lab_Chroma~~ and should be ignored for analysis

Correlation 0.95511

~~Cytoplasm_HorizGraySTD_~~
Highly correlated

This variable is highly correlated with ~~Cytoplasm_HorizGrayMean_Lab_Hue~~ and should be ignored for analysis

Correlation 0.93832

Cytoplasm_HorizGraySkew
Numeric

Distinct count	2652	**Mean**	204430
Unique (%)	100.0%	**Minimum**	-70885000
Missing (%)	0.0%	**Maximum**	116230000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

1e8

Toggle details

Cytoplasm_HorizGraySkew
Numeric

Distinct count	2653	**Mean**	8213.4
Unique (%)	100.0%	**Minimum**	-22276
Missing (%)	0.0%	**Maximum**	1584600
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_Kurtosis_Lab_C
Numeric

Distinct count	2652	**Mean**	43403000
Unique (%)	100.0%	**Minimum**	4.877e-05
Missing (%)	0.0%	**Maximum**	30496000000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

1e10

Toggle details

Cytoplasm_Kurtosis_Lab_H
Numeric

Distinct count	2652	**Mean**	106280
Unique (%)	100.0%	**Minimum**	1.1581e-07
Missing (%)	0.0%	**Maximum**	80076000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

1e8

Toggle details

~~Cytoplasm_L1Norm_Lab_CH~~
Highly correlated

This variable is highly correlated with `Cytoplasm_Entropy_Lab_Chroma` and should be ignored for analysis

Correlation 0.99762

~~Cytoplasm_L1Norm_Lab_Hu~~
Highly correlated

This variable is highly correlated with `Cytoplasm_Entropy_Lab_Hue` and should be ignored for analysis

Correlation 0.96361

~~Cytoplasm_L2Norm_Lab_CH~~
Highly correlated

This variable is highly correlated with `Cytoplasm_L1Norm_Lab_Chroma` and should be ignored for analysis

Correlation 0.99419

~~Cytoplasm_L2Norm_Lab_Hu~~
Highly correlated

This variable is highly correlated with `Cytoplasm_L1Norm_Lab_Hue`

Correlation 0.97197

and should be ignored for analysis

Cytoplasm_LaplacianMAX
Numeric

Distinct count	67	**Mean**	18.834
Unique (%)	2.5%	**Minimum**	6
Missing (%)	0.0%	**Maximum**	87
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_LaplacianMEAI
Numeric

Distinct count	2363	**Mean**	2.1429
Unique (%)	89.1%	**Minimum**	0.8238
Missing (%)	0.0%	**Maximum**	12.069
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_LaplacianSTD~~
Highly correlated

This variable is highly correlated with `Cytoplasm_LaplacianMEAN` and should be ignored for analysis

Correlation 0.94073

~~Cytoplasm_Mean_Lab_Chro~~
Highly correlated

This variable is highly correlated with `Cytoplasm_Energy_Lab_Chroma` and should be ignored for analysis

Correlation 0.99959

~~Cytoplasm_Mean_Lab_Hue~~
Highly correlated

This variable is highly correlated with `Cytoplasm_Energy_Lab_Hue` and should be ignored for analysis

Correlation 0.99716

Cytoplasm_STD_Lab_Chro
Numeric

Distinct count	2647	**Mean**	2.5846
Unique (%)	99.8%	**Minimum**	0.23294
Missing (%)	0.0%	**Maximum**	19.143
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_STD_Lab_Hue
Numeric

Distinct count	2645	**Mean**	1.675
Unique (%)	99.7%	**Minimum**	0.11565

Missing (%)	0.0%	Maximum	9.0945
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_Skewness_Lab
Numeric

Distinct count	2653	Mean	17163
Unique (%)	100.0%	Minimum	-2705000
Missing (%)	0.0%	Maximum	18346000
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_Skewness_Lab
Numeric

Distinct count	2652	Mean	53.12
Unique (%)	100.0%	Minimum	-62845
Missing (%)	0.0%	Maximum	104530
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_VertGrayEnergy
Numeric

Distinct count	2651	Mean	80.128
Unique (%)	99.9%	Minimum	0.40661
Missing (%)	0.0%	Maximum	4866.8
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_VertGrayEnergy
Numeric

Distinct count	2649	Mean	25.057
Unique (%)	99.8%	Minimum	0.078811
Missing (%)	0.0%	Maximum	691.63
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_VertGrayEntro~~
Highly correlated

This variable is highly correlated with Cytoplasm_VertGrayEnergy_Lab_Chroma and should be ignored for analysis

Correlation	0.92486

Cytoplasm_VertGrayEntrop
Numeric

Distinct count	2651	Mean	18771
Unique (%)	99.9%	Minimum	-17.089
Missing (%)	0.0%	Maximum	668680
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		

Infinite (n)	0		

Toggle details

Cytoplasm_VertGrayKurtos
Numeric

Distinct count	2653	Mean	1405100000
Unique (%)	100.0%	Minimum	0.01003
Missing (%)	0.0%	Maximum	1879000000000
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_VertGrayKurtos
Numeric

Distinct count	2653	Mean	2545700
Unique (%)	100.0%	Minimum	7.8466e-06
Missing (%)	0.0%	Maximum	1197400000
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_VertGrayL1Nor
Numeric

Distinct count	2650	Mean	701.91
Unique (%)	99.9%	Minimum	24.642
Missing (%)	0.0%	Maximum	8648.5
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_VertGrayL1Nor
Numeric

Distinct count	2651	Mean	482.67
Unique (%)	99.9%	Minimum	8.9976
Missing (%)	0.0%	Maximum	4635.1
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_VertGrayL2No~~
Highly correlated

This variable is highly correlated with Cytoplasm_VertGrayL1Norm_Lab_Chroma and should be ignored for analysis

Correlation	0.96511

~~Cytoplasm_VertGrayL2No~~
Highly correlated

This variable is highly correlated with Cytoplasm_VertGrayL1Norm_Lab_Hue and should be ignored for analysis

Correlation	0.96905

Cytoplasm_VertGrayMean_
Numeric

Distinct count	2649	**Mean**	5.3421
Unique (%)	99.8%	**Minimum**	0.34162
Missing (%)	0.0%	**Maximum**	62.219
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_VertGrayMean_~~
Highly correlated

This variable is highly correlated with Cytoplasm_VertGrayEnergy_Lab_Hue and should be ignored for analysis

Correlation 0.92395

~~Cytoplasm_VertGraySTD_L~~
Highly correlated

This variable is highly correlated with Cytoplasm_VertGrayMean_Lab_Chroma and should be ignored for analysis

Correlation 0.94961

~~Cytoplasm_VertGraySTD_L~~
Highly correlated

This variable is highly correlated with Cytoplasm_VertGrayMean_Lab_Hue and should be ignored for analysis

Correlation 0.9359

Cytoplasm_VertGraySkewr
Numeric

Distinct count	2651	**Mean**	152950
Unique (%)	99.9%	**Minimum**	-112900000
Missing (%)	0.0%	**Maximum**	102790000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_VertGraySkewr
Numeric

Distinct count	2653	**Mean**	7881.9
Unique (%)	100.0%	**Minimum**	-16208
Missing (%)	0.0%	**Maximum**	1513100
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobArea
Numeric

Distinct count	1979	**Mean**	9.6326
Unique (%)	74.6%	**Minimum**	5.0787
Missing (%)	0.0%	**Maximum**	20.02
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_blobAsymmetry~~
Highly correlated

This variable is highly correlated with Cytoplasm_blobMinorAsymmetryIndex and should be ignored for analysis

Correlation 0.98728

Cytoplasm_blobAsymmetry
Numeric

Distinct count	1356	**Mean**	0.827
Unique (%)	51.1%	**Minimum**	0.45033
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_blobAsymmetry~~
Highly correlated

This variable is highly correlated with Cytoplasm_blobAsymmetry_Celebi1 and should be ignored for analysis

Correlation 0.92159

~~Cytoplasm_blobBoundingE~~
Highly correlated

This variable is highly correlated with Cytoplasm_blobConvexHullArea and should be ignored for analysis

Correlation 0.90039

Cytoplasm_blobBoundingE
Numeric

Distinct count	563	**Mean**	1.1391
Unique (%)	21.2%	**Minimum**	0.16177
Missing (%)	0.0%	**Maximum**	6.8333
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_blobCircular~~
Highly correlated

This variable is highly correlated with Cytoplasm_blobSolidity and should be ignored for analysis

Correlation 0.90825

Cytoplasm_blobCompactn
Numeric

Distinct count	2503	**Mean**	1.9231
Unique (%)	94.3%	**Minimum**	0.89038
Missing (%)	0.0%	**Maximum**	7.7968
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobConvexHul
Highly correlated

This variable is highly correlated with Cytoplasm_blobPerimeter and should be ignored for analysis

Correlation 0.94681

Cytoplasm_blobEccentricit
Numeric

Distinct count	2650	**Mean**	8.2621
Unique (%)	99.9%	**Minimum**	1.0206
Missing (%)	0.0%	**Maximum**	130.8
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobEllipticalSy
Numeric

Distinct count	2265	**Mean**	0.25981
Unique (%)	85.4%	**Minimum**	0.0089286
Missing (%)	0.0%	**Maximum**	0.80123
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobElongation
Numeric

Distinct count	2648	**Mean**	0.35278
Unique (%)	99.8%	**Minimum**	0.0040864
Missing (%)	0.0%	**Maximum**	1.0751
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobEquivaler
Highly correlated

This variable is highly correlated with Cytoplasm_blobConvexHullArea and should be ignored for analysis

Correlation 0.90651

Cytoplasm_blobExtent
Numeric

Distinct count	2029	**Mean**	0.61667
Unique (%)	76.5%	**Minimum**	0.14914
Missing (%)	0.0%	**Maximum**	1.0833
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobIrregularity
Numeric

Distinct count	2502	**Mean**	0.36615
Unique (%)	94.3%	**Minimum**	0.19285
Missing (%)	0.0%	**Maximum**	0.61699
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobIrregula
Highly correlated

This variable is highly correlated with Cytoplasm_blobPrincipalAxisRatio and should be ignored for analysis

Correlation 0.97148

Cytoplasm_blobIrregula
Highly correlated

This variable is highly correlated with Cytoplasm_blobBoundingBoxRatio and should be ignored for analysis

Correlation 0.93237

Cytoplasm_blobIrregula
Highly correlated

This variable is highly correlated with Cytoplasm_blobIrregularityIndexC and should be ignored for analysis

Correlation 0.95221

Cytoplasm_blobLengtheni
Numeric

Distinct count	2651	**Mean**	0.27593
Unique (%)	99.9%	**Minimum**	0.0076455
Missing (%)	0.0%	**Maximum**	0.97986
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobMajorAsyn
Numeric

Distinct count	2061	**Mean**	0.42464
Unique (%)	77.7%	**Minimum**	0.022727
Missing (%)	0.0%	**Maximum**	0.98198
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobMajorAsyn
Numeric

Distinct count	1300	**Mean**	0.87176
Unique (%)	49.0%	**Minimum**	0.45033
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobMaxDiame
Numeric

Distinct count	901	**Mean**	0.96502
Unique (%)	34.0%	**Minimum**	0.25278
Missing (%)	0.0%	**Maximum**	3.2458
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobMinDiamet
Numeric

Distinct count	923	**Mean**	0.97354
Unique (%)	34.8%	**Minimum**	0.25536
Missing (%)	0.0%	**Maximum**	3.5895
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Cytoplasm_blobMinorASyn~~
Highly correlated

This variable is highly correlated with Cytoplasm_blobMajorAsymmetryIndex and should be ignored for analysis

Correlation	0.94962

Cytoplasm_blobMinorAsyn
Numeric

Distinct count	1285	**Mean**	0.87545
Unique (%)	48.4%	**Minimum**	0.45192
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobPerimeter
Numeric

Distinct count	2286	**Mean**	3.427
Unique (%)	86.2%	**Minimum**	1.7235
Missing (%)	0.0%	**Maximum**	10.831
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobPrincipalA:
Numeric

Distinct count	563	**Mean**	1.148
Unique (%)	21.2%	**Minimum**	0.14634
Missing (%)	0.0%	**Maximum**	6.1818
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobRadialVari
Numeric

Distinct count	2648	**Mean**	10.987
Unique (%)	99.8%	**Minimum**	0.86518
Missing (%)	0.0%	**Maximum**	113.29
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Cytoplasm_blobSolidity
Numeric

Distinct count	2183	**Mean**	0.83101
Unique (%)	82.3%	**Minimum**	0.38492
Missing (%)	0.0%	**Maximum**	1.1183
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Label
Numeric

Distinct count	2	**Mean**	1.8847
Unique (%)	0.1%	**Minimum**	1
Missing (%)	0.0%	**Maximum**	2
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Ratio_LAB_Chroma_Mean
Numeric

Distinct count	2652	**Mean**	-0.0018191
Unique (%)	100.0%	**Minimum**	-0.13033
Missing (%)	0.0%	**Maximum**	0.1285
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Ratio_LAB_Chroma_Mean2
Numeric

Distinct count	2650	**Mean**	6796.9
Unique (%)	99.9%	**Minimum**	-486.44
Missing (%)	0.0%	**Maximum**	11764000
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Ratio_LAB_Hue_Mean1
Numeric

Distinct count	2653	**Mean**	-0.016329
Unique (%)	100.0%	**Minimum**	-0.41483
Missing (%)	0.0%	**Maximum**	0.26242
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Ratio_LAB_Hue_Mean2
Numeric

Distinct count	2650	**Mean**	16447
Unique (%)	99.9%	**Minimum**	-219.95
Missing (%)	0.0%	**Maximum**	11387000
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Ratio_blobArea
Numeric

Distinct count	1688	**Mean**	0.12362
Unique (%)	63.6%	**Minimum**	0.021951
Missing (%)	0.0%	**Maximum**	0.57265
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Ratio_blobBoundingBoxAr~~
Highly correlated

This variable is highly correlated with `Ratio_blobConvexHullArea` and should be ignored for analysis

Correlation 0.94282

~~Ratio_blobConvexHullAre~~
Highly correlated

This variable is highly correlated with `Ratio_blobPerimeter` and should be ignored for analysis

Correlation 0.95667

Ratio_blobMaxDiameter
Numeric

Distinct count	206	**Mean**	0.26584
Unique (%)	7.8%	**Minimum**	0.048387
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Ratio_blobMinDiameter
Numeric

Distinct count	216	**Mean**	0.26228
Unique (%)	8.1%	**Minimum**	0.033898
Missing (%)	0.0%	**Maximum**	1
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

~~Ratio_blobPerimeter~~
Highly correlated

This variable is highly correlated with `Ratio_blobArea` and should be ignored for analysis

Correlation 0.90072

~~ID~~
Highly correlated

This variable is highly correlated with `index` and should be ignored for analysis

Correlation 1

index
Numeric

Distinct count	2653	**Mean**	1326
Unique (%)	100.0%	**Minimum**	0
Missing (%)	0.0%	**Maximum**	2652
Missing (n)	0	**Zeros (%)**	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

Toggle details

Sample

	ID	Chromatine_blobArea	Chromatine_blobMaxDiameter	Chromatine_blobMinDiameter	Chromatine_blobPerimeter	Chu
2470	2470	0.840336	0.148295	0.197726	0.576216	0.7
173	173	1.504700	0.250784	0.313480	0.944895	1.3
1229	1229	0.564103	0.205128	0.153846	0.537707	0.5
2140	2140	0.619835	0.103306	0.206612	0.498805	0.4
1199	1199	0.769231	0.256410	0.102564	0.597787	0.7

Appendix B

B.1 Final Results Table

Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 350})	0,776966568	0,027271411
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 450})	0,772232099	0,031419272
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 200})	0,772125845	0,032270519
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 400})	0,770960633	0,024562631
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 300})	0,770925643	0,024167871
{Dummy Selector} AdaBoost({'n_estimators': 400})	0,767043761	0,024376803
{Dummy Selector} AdaBoost({'n_estimators': 500})	0,766778217	0,028574031
{Dummy Selector} AdaBoost({'n_estimators': 200})	0,76648769	0,03059761
{Dummy Selector} AdaBoost({'n_estimators': 150})	0,765585596	0,025884891
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 500})	0,765130152	0,021919031
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 250})	0,765103459	0,028292915
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 100})	0,763924452	0,034375616
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 500})	0,76392199	0,032578563
{Dummy Selector} AdaBoost({'n_estimators': 350})	0,763375776	0,022969098
{Fdr} AdaBoost({'n_estimators': 200})	0,76311036	0,030546281
{Dummy Selector} AdaBoost({'n_estimators': 450})	0,762962604	0,02360556
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 450})	0,761850335	0,045653245
{Dummy Selector} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,760821092	0,048909144
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 200})	0,760098445	0,039384263
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 350})	0,75875883	0,034290023
{Fdr} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,75818885	0,038957063
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 250})	0,757238926	0,034511464
{Fdr} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,756783721	0,038571661
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 300})	0,756708331	0,039685922
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 400})	0,756698692	0,039151228
{Dummy Selector} AdaBoost({'n_estimators': 100})	0,756629766	0,032673988
{Fdr} AdaBoost({'n_estimators': 450})	0,75642667	0,02676372
{Dummy Selector} AdaBoost({'n_estimators': 300})	0,756011055	0,034647919
{Dummy Selector} AdaBoost({'n_estimators': 250})	0,755354089	0,042567674
{Dummy Selector} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,755181339	0,046728805
{Variance Threshold({'threshold': 0.10})} AdaBoost({'n_estimators': 150})	0,754199272	0,032081621
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 150})	0,75355835	0,039041901
{KBest({'k': 100})} AdaBoost({'n_estimators': 200})	0,753326172	0,033755963
{Dummy Selector} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,75280557	0,03543427
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 300})	0,752758407	0,038304972
{KBest({'k': 100})} AdaBoost({'n_estimators': 100})	0,752643473	0,03190386
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 450})	0,752143525	0,022104572
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 100})	0,752136141	0,044602566
{Fdr} AdaBoost({'n_estimators': 500})	0,752121381	0,037420122
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 400})	0,751953108	0,024705914
{Fdr} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,751394517	0,041388255
{Dummy Selector} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,751135904	0,035845312
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 500})	0,750931176	0,021753539
{Fdr} AdaBoost({'n_estimators': 150})	0,750408086	0,024676362
{Fdr} AdaBoost({'n_estimators': 350})	0,750313306	0,029084522
{Fdr} AdaBoost({'n_estimators': 100})	0,750122438	0,027579357
{Fdr} AdaBoost({'n_estimators': 250})	0,749977159	0,045871131
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,749350726	0,044340506
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 250})	0,749217342	0,047648985
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 350})	0,749024998	0,031041361
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,748803907	0,039320042
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 150})	0,748611479	0,032615384
{KBest({'k': 100})} AdaBoost({'n_estimators': 250})	0,748453578	0,036153071
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 150})	0,748297676	0,051176274

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Fdr} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,748094084	0,038415697
{Fdr} AdaBoost({'n_estimators': 400})	0,746728643	0,032363878
{Fdr} AdaBoost({'n_estimators': 300})	0,746496614	0,039538858
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 300})	0,746261982	0,035447573
{KBest({'k': 100})} AdaBoost({'n_estimators': 150})	0,74625102	0,02588265
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,746128119	0,032425111
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 250})	0,746059603	0,030571221
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,746040999	0,028352068
{Variance Threshold({'threshold': 0.05})} AdaBoost({'n_estimators': 100})	0,745845318	0,041956197
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 200})	0,7455181	0,038440692
{KBest({'k': 100})} AdaBoost({'n_estimators': 300})	0,74532233	0,037788765
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,745321649	0,043711131
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,745169613	0,033166646
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 100})	0,745101035	0,046542038
{KBest({'k': 80})} AdaBoost({'n_estimators': 250})	0,744178854	0,035742662
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 500})	0,743921236	0,039771642
{KBest({'k': 80})} AdaBoost({'n_estimators': 100})	0,743697122	0,032290253
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,742788231	0,032686416
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,742318584	0,048245647
{Dummy Selector} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,742302074	0,034389834
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,74226881	0,033166306
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,742199202	0,044816263
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,742043393	0,038890415
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,741991176	0,038423321
{KBest({'k': 90})} AdaBoost({'n_estimators': 250})	0,741875674	0,03753556
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,741811318	0,043907838
{KBest({'k': 90})} AdaBoost({'n_estimators': 100})	0,741746803	0,030513522
{KBest({'k': 80})} AdaBoost({'n_estimators': 350})	0,741562333	0,034209794
{Fdr} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,741229735	0,038266253
{KBest({'k': 80})} AdaBoost({'n_estimators': 300})	0,741117715	0,031402014
{KBest({'k': 90})} AdaBoost({'n_estimators': 200})	0,741050723	0,038899735
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,740659591	0,039389759
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 450})	0,740626441	0,029285307
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 350})	0,740526261	0,032601102
{Variance Threshold({'threshold': 0.15})} AdaBoost({'n_estimators': 200})	0,74047201	0,039891647
{KBest({'k': 90})} AdaBoost({'n_estimators': 400})	0,740372315	0,035767667
{KBest({'k': 100})} AdaBoost({'n_estimators': 350})	0,740139042	0,043384148
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,739947993	0,048503731
{Variance Threshold({'threshold': 0.25})} AdaBoost({'n_estimators': 400})	0,73915788	0,034216766
{KBest({'k': 80})} AdaBoost({'n_estimators': 200})	0,739022569	0,037932463
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,738600916	0,042546039
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,738202714	0,036839014
{KBest({'k': 100})} AdaBoost({'n_estimators': 500})	0,737820977	0,036436788
{KBest({'k': 90})} AdaBoost({'n_estimators': 450})	0,736740184	0,029565527
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,736737284	0,044111853
{KBest({'k': 80})} AdaBoost({'n_estimators': 150})	0,736642589	0,034623798
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,736540804	0,037626526
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,736257893	0,044645884
{KBest({'k': 100})} AdaBoost({'n_estimators': 450})	0,736214286	0,034896936
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,736208205	0,042919833
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,736155144	0,042029173
{KBest({'k': 80})} AdaBoost({'n_estimators': 500})	0,736138842	0,033012428
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,735987544	0,044592097
{Dummy Selector} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,735069066	0,039206694
{KBest({'k': 90})} AdaBoost({'n_estimators': 500})	0,734513732	0,033347075
{KBest({'k': 100})} AdaBoost({'n_estimators': 400})	0,73442111	0,036381163
{KBest({'k': 80})} AdaBoost({'n_estimators': 450})	0,734257809	0,033029322
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,734144694	0,03971388
{KBest({'k': 90})} AdaBoost({'n_estimators': 150})	0,733883719	0,033727672
{KBest({'k': 80})} AdaBoost({'n_estimators': 400})	0,733294645	0,037271889
{KBest({'k': 90})} AdaBoost({'n_estimators': 300})	0,733218087	0,036795888
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,732467316	0,045338617
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,732175774	0,045346771
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,731779065	0,037811968
{KBest({'k': 90})} AdaBoost({'n_estimators': 350})	0,731613446	0,035144432
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,731595835	0,042356595
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 100})	0,731082113	0,030157161
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 450})	0,729545282	0,03569345
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 350})	0,729253279	0,029580533
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,729050591	0,039106195
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 400})	0,728019151	0,029221299
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,727948015	0,038931693
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,727529182	0,041156206
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,727434273	0,037889469
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 100})	0,726953837	0,038602445
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 500})	0,726616005	0,034534827

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,726323421	0,031446375
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,726323421	0,031446375
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 300})	0,726236128	0,030812087
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,725955478	0,030808973
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 250})	0,725660687	0,032073448
{Fdr} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,725327343	0,033408803
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 200})	0,724579848	0,038941627
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 150})	0,724464983	0,032767157
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,724045883	0,031171867
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 400})	0,723844914	0,040014678
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 300})	0,72351402	0,039645548
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,723507492	0,03604828
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,72286296	0,046971045
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 100})	0,722515941	0,042576337
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 150})	0,722370056	0,034420543
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 300})	0,722335804	0,041069955
{Fdr} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,722334526	0,035464523
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,722284203	0,036947796
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,722010432	0,032895988
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,721781031	0,050211749
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 200})	0,721446198	0,027901863
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 300})	0,72142035	0,041507447
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 350})	0,721273982	0,041530041
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 500})	0,721237712	0,040817161
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 700})	0,721212572	0,02669719
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,720686464	0,043719209
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,72037286	0,040475313
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 350})	0,720079435	0,045279422
{Fdr} Extremely Randomized Trees({'n_estimators': 250})	0,719890916	0,035031481
{Fdr} Extremely Randomized Trees({'n_estimators': 350})	0,719864586	0,038885951
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,719599396	0,04738697
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 200})	0,719466471	0,031667597
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 500})	0,71944034	0,037209769
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 500})	0,719370746	0,026853246
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 600})	0,719370746	0,026853246
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 300})	0,719353672	0,047290328
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 350})	0,719286723	0,036992902
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,719203373	0,047583299
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 300})	0,719174631	0,026773531
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 900})	0,719099487	0,031943155
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 1000})	0,719099487	0,031943155
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 800})	0,71901151	0,033338286
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,718975071	0,041041119
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 50})	0,718888315	0,032010811
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 400})	0,718859488	0,041031605
{Fdr} Extremely Randomized Trees({'n_estimators': 400})	0,718592447	0,039393285
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 400})	0,718590894	0,045975249
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 50})	0,718520102	0,040967841
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 100})	0,718242635	0,034298197
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 450})	0,71823858	0,041098455
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 450})	0,718166753	0,044743071
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 450})	0,718111731	0,042277881
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 250})	0,717602977	0,045860301
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 800})	0,717500448	0,026355095
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 1000})	0,717500448	0,026355095
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 300})	0,717137952	0,029167491
{Fdr} Random Forest({'n_estimators': 400})	0,71692917	0,030958362
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 300})	0,716838218	0,026511113
{Fdr} Extremely Randomized Trees({'n_estimators': 450})	0,716781864	0,038874529
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 600})	0,716753285	0,032745931
{Fdr} Random Forest({'n_estimators': 200})	0,716610195	0,029515991
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 400})	0,716552365	0,026485959
{Dummy Selector} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,716385103	0,034645748
{Variance Threshold({'threshold': 0.10})} Extremely Randomized Trees({'n_estimators': 250})	0,716363105	0,041420298
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 500})	0,716242993	0,034042275
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 400})	0,715998564	0,036068454
{Fdr} Random Forest({'n_estimators': 900})	0,715914963	0,025252945
{Fdr} Random Forest({'n_estimators': 1000})	0,715914963	0,025252945
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 200})	0,715792518	0,030484892
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 900})	0,715575204	0,027725075
{Dummy Selector} Random Forest({'n_estimators': 400})	0,715541519	0,026826277
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 400})	0,715343641	0,040810462
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 200})	0,715331185	0,042691704
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 700})	0,715269466	0,034702595
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 350})	0,715209849	0,045219203
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 100})	0,715153141	0,047195127

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 450})	0,715128352	0,042585469
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 250})	0,715062448	0,040220344
{KBest({'k': 20})} Random Forest({'n_estimators': 900})	0,71496865	0,039520498
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 800})	0,714964864	0,033368814
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 1000})	0,714964864	0,033368814
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 500})	0,714773738	0,048276921
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 500})	0,714739309	0,035134126
{Fdr} Extremely Randomized Trees({'n_estimators': 200})	0,714723457	0,036446295
{KBest({'k': 90})} Random Forest({'n_estimators': 400})	0,714671125	0,035934196
{Variance Threshold({'threshold': 0.05})} Extremely Randomized Trees({'n_estimators': 50})	0,714638513	0,036272381
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 500})	0,714541511	0,037294895
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 600})	0,714526119	0,033611486
{Dummy Selector} Random Forest({'n_estimators': 600})	0,71444944	0,029562548
{Dummy Selector} Random Forest({'n_estimators': 800})	0,714414917	0,028580329
{KBest({'k': 100})} Random Forest({'n_estimators': 500})	0,714293717	0,035292706
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 1000})	0,714236176	0,030590766
{KBest({'k': 20})} Random Forest({'n_estimators': 1000})	0,714147861	0,035746957
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 400})	0,714144398	0,032364686
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,713961283	0,053960994
{Fdr} Extremely Randomized Trees({'n_estimators': 300})	0,713652318	0,037898999
{Fdr} Extremely Randomized Trees({'n_estimators': 100})	0,7136454	0,037303617
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 400})	0,713546947	0,031628844
{Variance Threshold({'threshold': 0.10})} Random Forest({'n_estimators': 200})	0,713487906	0,032708864
{Fdr} Random Forest({'n_estimators': 700})	0,713372747	0,026612163
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 250})	0,713162984	0,046962947
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 500})	0,71312503	0,046339128
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,713120495	0,040062295
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 800})	0,713078143	0,030642006
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 900})	0,713078143	0,030642006
{Fdr} Extremely Randomized Trees({'n_estimators': 500})	0,713061064	0,038239956
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 900})	0,71303962	0,03430333
{KBest({'k': 20})} Random Forest({'n_estimators': 300})	0,713032232	0,039163925
{Dummy Selector} Random Forest({'n_estimators': 700})	0,712929431	0,033824991
{Fdr} Random Forest({'n_estimators': 800})	0,712778141	0,027597839
{Dummy Selector} Random Forest({'n_estimators': 100})	0,71267565	0,024631666
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,712612455	0,030503661
{KBest({'k': 20})} Random Forest({'n_estimators': 500})	0,712575928	0,039062596
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 150})	0,712535155	0,046673688
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 700})	0,712471468	0,029083987
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,71245536	0,045701754
{KBest({'k': 20})} Random Forest({'n_estimators': 800})	0,71240742	0,040425873
{Dummy Selector} Random Forest({'n_estimators': 200})	0,712332276	0,02746084
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 200})	0,71232932	0,043423311
{KBest({'k': 20})} Random Forest({'n_estimators': 400})	0,712275818	0,03616172
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 200})	0,712272969	0,040462283
{KBest({'k': 20})} Random Forest({'n_estimators': 600})	0,712244034	0,037268784
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 250})	0,712235199	0,040242649
{Variance Threshold({'threshold': 0.15})} Extremely Randomized Trees({'n_estimators': 150})	0,71220966	0,038776638
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 150})	0,712206405	0,041693885
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 200})	0,712131009	0,041570244
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,712019744	0,042839032
{KBest({'k': 20})} Random Forest({'n_estimators': 200})	0,711984375	0,0430025
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 150})	0,711912978	0,041746188
{KBest({'k': 100})} Random Forest({'n_estimators': 400})	0,711781349	0,038105838
{KBest({'k': 90})} Random Forest({'n_estimators': 500})	0,711780086	0,033628281
{Fdr} Random Forest({'n_estimators': 500})	0,711742959	0,032863741
{KBest({'k': 100})} Random Forest({'n_estimators': 900})	0,711711548	0,036789113
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 100})	0,711647274	0,046563761
{KBest({'k': 100})} Random Forest({'n_estimators': 100})	0,711636739	0,043131009
{Fdr} Extremely Randomized Trees({'n_estimators': 150})	0,711531222	0,038243424
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,71133322	0,044979846
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 400})	0,711327098	0,035562767
{Dummy Selector} Random Forest({'n_estimators': 500})	0,71121196	0,032285352
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 200})	0,711205192	0,042524676
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,71116321	0,043307102
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 150})	0,711148604	0,035553541
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 700})	0,711025644	0,036362188
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 100})	0,71098972	0,03736638
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 100})	0,710938014	0,042422249
{KBest({'k': 10})} Random Forest({'n_estimators': 100})	0,710818688	0,044790451
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 50})	0,710377359	0,050609112
{KBest({'k': 100})} Random Forest({'n_estimators': 300})	0,710338832	0,038079517
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 50})	0,710323378	0,04946772
{Dummy Selector} Random Forest({'n_estimators': 1000})	0,710293049	0,03481718
{KBest({'k': 100})} Random Forest({'n_estimators': 1000})	0,710282807	0,037736704
{Fdr} Random Forest({'n_estimators': 600})	0,71027947	0,027439109

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 90})} Random Forest({'n_estimators': 700})	0,710262349	0,037960568
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 600})	0,71005266	0,02991201
{Dummy Selector} Random Forest({'n_estimators': 300})	0,709925894	0,026332996
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,70987682	0,041686793
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 300})	0,70949104	0,047206904
{KBest({'k': 80})} Random Forest({'n_estimators': 400})	0,709398471	0,048751938
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 250})	0,709310323	0,042489669
{KBest({'k': 20})} Random Forest({'n_estimators': 700})	0,709209404	0,036597752
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 300})	0,70911611	0,038527109
{KBest({'k': 90})} Random Forest({'n_estimators': 600})	0,709075835	0,037300848
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 450})	0,70901524	0,042487911
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 50})	0,709014009	0,035310454
{KBest({'k': 100})} Random Forest({'n_estimators': 800})	0,708989112	0,037342522
{KBest({'k': 90})} Random Forest({'n_estimators': 900})	0,708968181	0,037993229
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 450})	0,708839645	0,037354769
{Dummy Selector} Random Forest({'n_estimators': 900})	0,708788976	0,035085115
{Fdr} Random Forest({'n_estimators': 300})	0,708779728	0,028529181
{Variance Threshold({'threshold': 0.25})} Random Forest({'n_estimators': 500})	0,708776412	0,027823005
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 400})	0,708743948	0,049257425
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 200})	0,708727059	0,049163694
{KBest({'k': 90})} Random Forest({'n_estimators': 300})	0,708713209	0,040740917
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 300})	0,708675609	0,045444943
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 300})	0,708641461	0,032170903
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 500})	0,708578404	0,037425483
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 150})	0,708521069	0,050811903
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,708347103	0,045201834
{KBest({'k': 90})} Random Forest({'n_estimators': 100})	0,708298656	0,036010192
{Fdr} Extremely Randomized Trees({'n_estimators': 50})	0,708208332	0,04032893
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,708155911	0,026186372
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 350})	0,708076039	0,042944648
{KBest({'k': 10})} Random Forest({'n_estimators': 400})	0,70806374	0,038893867
{KBest({'k': 100})} Random Forest({'n_estimators': 700})	0,708033473	0,036135179
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 450})	0,707922325	0,037601768
{KBest({'k': 10})} Random Forest({'n_estimators': 1000})	0,707820904	0,036856562
{Fdr} Random Forest({'n_estimators': 100})	0,707805369	0,028758278
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 400})	0,707747633	0,041528516
{KBest({'k': 90})} Random Forest({'n_estimators': 1000})	0,707735335	0,038318443
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 500})	0,707343381	0,042388841
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 100})	0,707226337	0,052738506
{KBest({'k': 90})} Random Forest({'n_estimators': 200})	0,707081609	0,043666255
{KBest({'k': 90})} Random Forest({'n_estimators': 800})	0,70707392	0,038804985
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 200})	0,707071182	0,036189865
{KBest({'k': 80})} Random Forest({'n_estimators': 100})	0,70691077	0,045638105
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 100})	0,706883505	0,038486672
{KBest({'k': 10})} Random Forest({'n_estimators': 500})	0,706776725	0,037966191
{KBest({'k': 80})} Random Forest({'n_estimators': 900})	0,706672921	0,04883844
{KBest({'k': 80})} Random Forest({'n_estimators': 700})	0,706588732	0,047911259
{KBest({'k': 10})} Random Forest({'n_estimators': 600})	0,706431722	0,038209308
{KBest({'k': 80})} Random Forest({'n_estimators': 1000})	0,70600914	0,048571732
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 350})	0,706007053	0,045664422
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 250})	0,705963349	0,044723423
{KBest({'k': 10})} Random Forest({'n_estimators': 900})	0,705908991	0,039195472
{KBest({'k': 80})} Random Forest({'n_estimators': 500})	0,705690488	0,04521314
{KBest({'k': 80})} Random Forest({'n_estimators': 600})	0,705690488	0,04521314
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 50})	0,705676256	0,041996937
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 450})	0,705324434	0,043640647
{KBest({'k': 20})} AdaBoost({'n_estimators': 100})	0,705273599	0,027012031
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 350})	0,705270671	0,046350097
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 250})	0,705234833	0,048629804
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 450})	0,705221802	0,05001784
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 350})	0,705214571	0,048594388
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 300})	0,705013051	0,049030977
{KBest({'k': 100})} Random Forest({'n_estimators': 600})	0,704851236	0,037057534
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 250})	0,704760384	0,035648122
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 400})	0,704718405	0,047472126
{KBest({'k': 20})} Extremely Randomized Trees({'n_estimators': 500})	0,70460544	0,050339286
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 100})	0,704485406	0,045691674
{Variance Threshold({'threshold': 0.05})} Random Forest({'n_estimators': 100})	0,704481134	0,032762517
{KBest({'k': 10})} Random Forest({'n_estimators': 800})	0,704234362	0,040183348
{KBest({'k': 10})} Random Forest({'n_estimators': 700})	0,704166082	0,037945962
{KBest({'k': 10})} Random Forest({'n_estimators': 300})	0,704141802	0,036345898
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 400})	0,704043554	0,042936777
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 500})	0,703870862	0,041231118
{KBest({'k': 100})} Random Forest({'n_estimators': 200})	0,703709971	0,033243275
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 150})	0,703660507	0,039327475
{KBest({'k': 20})} Random Forest({'n_estimators': 100})	0,7033449	0,042539036

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.25})} Extremely Randomized Trees({'n_estimators': 100})	0,703186488	0,04514562
{KBest({'k': 80})} Random Forest({'n_estimators': 800})	0,702689244	0,047656102
{KBest({'k': 10})} Random Forest({'n_estimators': 200})	0,702616563	0,035941731
{KBest({'k': 80})} Random Forest({'n_estimators': 300})	0,702346997	0,048459263
{KBest({'k': 90})} Extremely Randomized Trees({'n_estimators': 50})	0,702183267	0,046602576
{Variance Threshold({'threshold': 0.15})} Random Forest({'n_estimators': 200})	0,701933057	0,039126897
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 350. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,701859853	0,035932206
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 450. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,701819866	0,035176172
{KBest({'k': 100})} Extremely Randomized Trees({'n_estimators': 300})	0,701797086	0,042053553
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 350})	0,701751688	0,04642063
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 500. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,70165933	0,037196814
{KBest({'k': 80})} Extremely Randomized Trees({'n_estimators': 50})	0,700759443	0,034522692
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 200})	0,700117879	0,045634834
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 400. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,699488623	0,03421263
{KBest({'k': 20})} AdaBoost({'n_estimators': 250})	0,699050984	0,037765611
{KBest({'k': 20})} AdaBoost({'n_estimators': 150})	0,698423137	0,037732591
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 100})	0,697970705	0,037787786
{KBest({'k': 80})} Random Forest({'n_estimators': 200})	0,697753386	0,046569633
{KBest({'k': 10})} Extremely Randomized Trees({'n_estimators': 150})	0,697693823	0,043633348
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 300. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,696675157	0,029523224
{Fdr} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,696188671	0,047292891
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 250. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,695595091	0,036496461
{KBest({'k': 20})} AdaBoost({'n_estimators': 300})	0,695476269	0,046030033
{KBest({'k': 20})} AdaBoost({'n_estimators': 200})	0,695077851	0,048643827
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,694728137	0,044836515
{KBest({'k': 10})} AdaBoost({'n_estimators': 100})	0,694668598	0,036045389
{Dummy Selector} Extremely Randomized Trees({'n_estimators': 50})	0,693655381	0,044220294
{Dummy Selector} Decision Tree	0,693559607	0,042218432
{KBest({'k': 20})} AdaBoost({'n_estimators': 350})	0,692915596	0,044046531
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,691880626	0,050505804
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,691488951	0,049490086
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,691016436	0,05126504
{KBest({'k': 20})} AdaBoost({'n_estimators': 450})	0,690211445	0,041078043
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,689973578	0,052317512
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,689494469	0,039259275
{KBest({'k': 20})} AdaBoost({'n_estimators': 500})	0,687068051	0,042202748
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 200. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,686979095	0,044439733
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,686977002	0,047186936
{KBest({'k': 20})} AdaBoost({'n_estimators': 400})	0,686503599	0,044284182
{KBest({'k': 10})} AdaBoost({'n_estimators': 400})	0,685886506	0,038401754
{KBest({'k': 10})} AdaBoost({'n_estimators': 150})	0,685251491	0,037766403
{KBest({'k': 10})} AdaBoost({'n_estimators': 250})	0,684533743	0,033530724
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,683892869	0,051835294
{Dummy Selector} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,683826486	0,050404211
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 150. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,681750584	0,032212469
{KBest({'k': 10})} AdaBoost({'n_estimators': 350})	0,681724921	0,039425881
{KBest({'k': 10})} AdaBoost({'n_estimators': 200})	0,679883012	0,039080353
{KBest({'k': 10})} AdaBoost({'n_estimators': 300})	0,679596792	0,039172965
{KBest({'k': 10})} AdaBoost({'n_estimators': 450})	0,678893797	0,037475743
{KBest({'k': 90})} Naive Bayes	0,67740902	0,025168959
{KBest({'k': 10})} AdaBoost({'n_estimators': 500})	0,677116537	0,040948843
{KBest({'k': 100})} Decision Tree	0,676228273	0,038327872
{KBest({'k': 100})} Naive Bayes	0,675070039	0,031626246
{Fdr} Decision Tree	0,666745574	0,028117311
{Variance Threshold({'threshold': 0.10})} Decision Tree	0,663307534	0,031736602
{Variance Threshold({'threshold': 0.25})} Decision Tree	0,659094511	0,026757706
{KBest({'k': 90})} Decision Tree	0,658765609	0,032943963
{KBest({'k': 100})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,658046164	0,050451084
{Variance Threshold({'threshold': 0.05})} Decision Tree	0,652682726	0,035864306
{KBest({'k': 80})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,6519444	0,052425251
{Variance Threshold({'threshold': 0.15})} Decision Tree	0,64904803	0,039746778
{KBest({'k': 10})} Decision Tree	0,648000886	0,041341907
{KBest({'k': 80})} Decision Tree	0,647816101	0,031042414
{KBest({'k': 90})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,645302312	0,055699464
{Fdr} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,643797837	0,068593863
{KBest({'k': 80})} Naive Bayes	0,6432564	0,02596268
{KBest({'k': 10})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,640741394	0,042782657
{Variance Threshold({'threshold': 0.10})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,638541752	0,051548886
{Variance Threshold({'threshold': 0.05})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,636579551	0,053889302
{KBest({'k': 10})} Naive Bayes	0,63644501	0,031897807
{KBest({'k': 20})} Decision Tree	0,635714858	0,030310374
{Dummy Selector} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,635014524	0,061988284
{KBest({'k': 20})} Naive Bayes	0,634429029	0,030057831
{KBest({'k': 20})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,634137002	0,037016756
{Variance Threshold({'threshold': 0.25})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,625992039	0,062067536
{Variance Threshold({'threshold': 0.15})} Gradient Boosting({'n_estimators': 100. 'max_depth': 4. 'min_samples_split': 2. 'learning_rate': 0.01})	0,62020668	0,048216974

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 80})} KNN({'n_neighbors': 3})	0,619446506	0,032945124
{KBest({'k': 100})} KNN({'n_neighbors': 3})	0,619313153	0,034167078
{KBest({'k': 20})} KNN({'n_neighbors': 3})	0,619058129	0,032904499
{KBest({'k': 90})} KNN({'n_neighbors': 3})	0,618411298	0,033546172
{KBest({'k': 100})} KNN({'n_neighbors': 5})	0,609727858	0,047543194
{KBest({'k': 90})} KNN({'n_neighbors': 5})	0,609727858	0,047543194
{Fdr} KNN({'n_neighbors': 1})	0,608692031	0,04373353
{KBest({'k': 100})} KNN({'n_neighbors': 1})	0,608125105	0,039233735
{KBest({'k': 90})} KNN({'n_neighbors': 1})	0,608094356	0,039264291
{Fdr} KNN({'n_neighbors': 3})	0,607480412	0,036904092
{KBest({'k': 80})} KNN({'n_neighbors': 5})	0,605952009	0,046789929
{KBest({'k': 20})} KNN({'n_neighbors': 5})	0,605952009	0,046789929
{KBest({'k': 20})} KNN({'n_neighbors': 1})	0,603559578	0,0353215
{KBest({'k': 80})} KNN({'n_neighbors': 1})	0,603192416	0,035027561
{Fdr} KNN({'n_neighbors': 5})	0,59178186	0,042596709
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 1})	0,58630294	0,063734378
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 1})	0,58630294	0,063734378
{Dummy Selector} KNN({'n_neighbors': 1})	0,58630294	0,063734378
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 1})	0,58630294	0,063734378
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 1})	0,58630294	0,063734378
{KBest({'k': 10})} KNN({'n_neighbors': 3})	0,581766641	0,029857587
{Fdr} KNN({'n_neighbors': 7})	0,577171418	0,042469322
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 3})	0,57547105	0,055120425
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 3})	0,57547105	0,055120425
{Dummy Selector} KNN({'n_neighbors': 3})	0,57547105	0,055120425
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 3})	0,57547105	0,055120425
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 3})	0,57547105	0,055120425
{KBest({'k': 80})} KNN({'n_neighbors': 2})	0,575339307	0,055075578
{KBest({'k': 20})} KNN({'n_neighbors': 2})	0,575339307	0,055075578
{KBest({'k': 90})} KNN({'n_neighbors': 7})	0,574466477	0,048343193
{KBest({'k': 100})} KNN({'n_neighbors': 7})	0,574088773	0,048211836
{KBest({'k': 100})} KNN({'n_neighbors': 2})	0,572634054	0,054054613
{KBest({'k': 90})} KNN({'n_neighbors': 2})	0,572497734	0,053639655
{Fdr} KNN({'n_neighbors': 2})	0,571797861	0,035213583
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,569708105	0,068612586
{Fdr} KNN({'n_neighbors': 9})	0,56965983	0,044952945
{KBest({'k': 80})} KNN({'n_neighbors': 4})	0,569417719	0,037527079
{KBest({'k': 20})} KNN({'n_neighbors': 4})	0,569417719	0,037527079
{KBest({'k': 100})} KNN({'n_neighbors': 4})	0,569281932	0,034322936
{KBest({'k': 90})} KNN({'n_neighbors': 4})	0,569281932	0,034322936
{KBest({'k': 80})} KNN({'n_neighbors': 7})	0,569276527	0,054687986
{KBest({'k': 20})} KNN({'n_neighbors': 7})	0,569276527	0,054687986
{Fdr} KNN({'n_neighbors': 4})	0,568548976	0,044994132
{KBest({'k': 100})} KNN({'n_neighbors': 9})	0,56572867	0,046312139
{KBest({'k': 90})} KNN({'n_neighbors': 9})	0,56572867	0,046312139
{KBest({'k': 80})} KNN({'n_neighbors': 9})	0,56572867	0,046312139
{KBest({'k': 20})} KNN({'n_neighbors': 9})	0,56572867	0,046312139
{KBest({'k': 10})} KNN({'n_neighbors': 1})	0,56225225	0,023914434
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 5})	0,558972694	0,049898958
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 5})	0,558972694	0,049898958
{Dummy Selector} KNN({'n_neighbors': 5})	0,558972694	0,049898958
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 5})	0,558972694	0,049898958
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 5})	0,558972694	0,049898958
{KBest({'k': 100})} KNN({'n_neighbors': 6})	0,558219643	0,046796009
{KBest({'k': 80})} KNN({'n_neighbors': 6})	0,555694794	0,049657787
{KBest({'k': 20})} KNN({'n_neighbors': 6})	0,555694794	0,049657787
{KBest({'k': 90})} KNN({'n_neighbors': 6})	0,555427117	0,049845379
{KBest({'k': 10})} KNN({'n_neighbors': 5})	0,553683374	0,04045123
{Variance Threshold({'threshold': 0.10})} Perceptron({'n_iter': 15})	0,55073897	0,05776328
{Variance Threshold({'threshold': 0.25})} Perceptron({'n_iter': 15})	0,55073897	0,05776328
{Dummy Selector} Perceptron({'n_iter': 15})	0,55073897	0,05776328
{Variance Threshold({'threshold': 0.15})} Perceptron({'n_iter': 15})	0,55073897	0,05776328
{Variance Threshold({'threshold': 0.05})} Perceptron({'n_iter': 15})	0,55073897	0,05776328
{Fdr} KNN({'n_neighbors': 6})	0,550013567	0,047051096
{KBest({'k': 10})} KNN({'n_neighbors': 7})	0,544192885	0,037696646
{Fdr} KNN({'n_neighbors': 8})	0,543056693	0,040292878
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 4})	0,54167734	0,040713999
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 4})	0,54167734	0,040713999
{Dummy Selector} KNN({'n_neighbors': 4})	0,54167734	0,040713999
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 4})	0,54167734	0,040713999
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 4})	0,54167734	0,040713999
{KBest({'k': 100})} KNN({'n_neighbors': 10})	0,541506873	0,042708812
{KBest({'k': 90})} KNN({'n_neighbors': 10})	0,541506873	0,042708812
{KBest({'k': 80})} KNN({'n_neighbors': 10})	0,541506873	0,042708812
{KBest({'k': 20})} KNN({'n_neighbors': 10})	0,541506873	0,042708812
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 9})	0,539986472	0,050231054

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 9})	0,539986472	0,050231054
{Dummy Selector} KNN({'n_neighbors': 9})	0,539986472	0,050231054
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 9})	0,539986472	0,050231054
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 9})	0,539986472	0,050231054
{KBest({'k': 80})} KNN({'n_neighbors': 8})	0,539309566	0,035199418
{KBest({'k': 20})} KNN({'n_neighbors': 8})	0,539309566	0,035199418
{KBest({'k': 90})} KNN({'n_neighbors': 8})	0,538987712	0,035110357
{KBest({'k': 100})} KNN({'n_neighbors': 8})	0,538677293	0,035049653
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 7})	0,538241746	0,051447874
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 7})	0,538241746	0,051447874
{Dummy Selector} KNN({'n_neighbors': 7})	0,538241746	0,051447874
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 7})	0,538241746	0,051447874
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 7})	0,538241746	0,051447874
{KBest({'k': 10})} KNN({'n_neighbors': 4})	0,534850883	0,04787225
{Variance Threshold({'threshold': 0.10})} Perceptron({'n_iter': 5})	0,532748618	0,040879909
{Variance Threshold({'threshold': 0.25})} Perceptron({'n_iter': 5})	0,532748618	0,040879909
{Dummy Selector} Perceptron({'n_iter': 5})	0,532748618	0,040879909
{Variance Threshold({'threshold': 0.15})} Perceptron({'n_iter': 5})	0,532748618	0,040879909
{Variance Threshold({'threshold': 0.05})} Perceptron({'n_iter': 5})	0,532748618	0,040879909
{Variance Threshold({'threshold': 0.10})} Perceptron({'n_iter': 10})	0,532612268	0,073162053
{Variance Threshold({'threshold': 0.25})} Perceptron({'n_iter': 10})	0,532612268	0,073162053
{Dummy Selector} Perceptron({'n_iter': 10})	0,532612268	0,073162053
{Variance Threshold({'threshold': 0.15})} Perceptron({'n_iter': 10})	0,532612268	0,073162053
{Variance Threshold({'threshold': 0.05})} Perceptron({'n_iter': 10})	0,532612268	0,073162053
{Fdr} Naive Bayes	0,530725728	0,030198678
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 2})	0,529356331	0,035318321
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 2})	0,529356331	0,035318321
{Dummy Selector} KNN({'n_neighbors': 2})	0,529356331	0,035318321
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 2})	0,529356331	0,035318321
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 2})	0,529356331	0,035318321
{Fdr} KNN({'n_neighbors': 10})	0,529345157	0,040881375
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,529240685	0,044641457
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,526091804	0,040762634
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 6})	0,5247629	0,043805513
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 6})	0,5247629	0,043805513
{Dummy Selector} KNN({'n_neighbors': 6})	0,5247629	0,043805513
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 6})	0,5247629	0,043805513
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 6})	0,5247629	0,043805513
{KBest({'k': 10})} KNN({'n_neighbors': 6})	0,520264043	0,030244021
{Variance Threshold({'threshold': 0.10})} Perceptron({'n_iter': 20})	0,518859639	0,095893508
{Variance Threshold({'threshold': 0.25})} Perceptron({'n_iter': 20})	0,518859639	0,095893508
{Dummy Selector} Perceptron({'n_iter': 20})	0,518859639	0,095893508
{Variance Threshold({'threshold': 0.15})} Perceptron({'n_iter': 20})	0,518859639	0,095893508
{Variance Threshold({'threshold': 0.05})} Perceptron({'n_iter': 20})	0,518859639	0,095893508
{KBest({'k': 10})} KNN({'n_neighbors': 9})	0,515969441	0,030688737
{KBest({'k': 10})} KNN({'n_neighbors': 2})	0,5155346	0,032464543
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 8})	0,510931357	0,034107672
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 8})	0,510931357	0,034107672
{Dummy Selector} KNN({'n_neighbors': 8})	0,510931357	0,034107672
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 8})	0,510931357	0,034107672
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 8})	0,510931357	0,034107672
{KBest({'k': 10})} KNN({'n_neighbors': 10})	0,509603752	0,022130519
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,508706465	0,036249081
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,508679821	0,035095153
{Variance Threshold({'threshold': 0.10})} KNN({'n_neighbors': 10})	0,50723015	0,032711181
{Variance Threshold({'threshold': 0.25})} KNN({'n_neighbors': 10})	0,50723015	0,032711181
{Dummy Selector} KNN({'n_neighbors': 10})	0,50723015	0,032711181
{Variance Threshold({'threshold': 0.15})} KNN({'n_neighbors': 10})	0,50723015	0,032711181
{Variance Threshold({'threshold': 0.05})} KNN({'n_neighbors': 10})	0,50723015	0,032711181
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,507138598	0,043313267
{KBest({'k': 10})} KNN({'n_neighbors': 8})	0,505217396	0,027716918
{Variance Threshold({'threshold': 0.10})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{Variance Threshold({'threshold': 0.25})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{Dummy Selector} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{KBest({'k': 100})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{KBest({'k': 10})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{KBest({'k': 90})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{Fdr} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{KBest({'k': 80})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{Variance Threshold({'threshold': 0.15})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{Variance Threshold({'threshold': 0.05})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{KBest({'k': 20})} DummyClassifier({'strategy': 'stratified'})	0,502761718	0,029652483
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,50248584	0,048242912
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,499094398	0,027237493
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,494982737	0,036530335
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,494960319	0,040332802

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,489414959	0,028315262
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,489104427	0,037512998
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,482208008	0,161730626
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,479373962	0,028220379
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,475707569	0,01349313
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,474050969	0,01334416
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,473692582	0,009766662
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,473306879	0,017295038
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,472722017	0,007502062
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,471955021	0,010337521
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,471559284	0,0090975
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,470725408	0,139504854
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,469877775	0,031848928
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,46971458	0,01063941
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.09000000000000011})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.6150000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.7400000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.8650000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 0.9900000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} Support Vector Machine	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.10})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.09000000000000011})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} Support Vector Machine	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.25})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{Dummy Selector} Support Vector Machine	0,469482745	0,000445087
{Dummy Selector} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,469482745	0,000445087

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,469482745	0,000445087
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.09000000000000011})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{KBest({'k': 100})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{KBest({'k': 100})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,469482745	0,000445087
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.09000000000000011})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 10})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{KBest({'k': 10})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{KBest({'k': 10})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,469482745	0,000445087
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{KBest({'k': 90})} Support Vector Machine	0,469482745	0,000445087
{KBest({'k': 90})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,469482745	0,000445087
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,469482745	0,000445087
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Fdr} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.6150000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.7400000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.8650000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 0.9900000000000001})	0,469482745	0,000445087
{Fdr} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{Fdr} Support Vector Machine	0,469482745	0,000445087
{Fdr} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,469482745	0,000445087
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,469482745	0,000445087
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,469482745	0,000445087
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,469482745	0,000445087
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.6150000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 80})} Support Vector Machine({'C': 0.7400000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.8650000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 0.9900000000000001})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{KBest({'k': 80})} Support Vector Machine	0,469482745	0,000445087
{KBest({'k': 80})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.6150000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.7400000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.8650000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 0.9900000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} Support Vector Machine	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.15})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Support Vector Machine	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,469482745	0,000445087
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,469482745	0,000445087
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,469482745	0,000445087
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.014999999999999999})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.040})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.065})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.090000000000000011})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.115})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.14000000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.16500000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.19})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.21500000000000002})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.23999999999999999})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.26500000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.29000000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.31500000000000006})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.34000000000000002})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.36500000000000005})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.39000000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.41500000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.44000000000000006})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.46500000000000002})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.49000000000000005})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.51500000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.54000000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.56500000000000006})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.59000000000000008})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.61500000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.64000000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.66500000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.69000000000000006})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.71500000000000008})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.74000000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.76500000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.79000000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.81500000000000006})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.84000000000000008})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.86500000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.89000000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.91500000000000004})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.94000000000000006})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.96500000000000008})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 0.99000000000000001})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine({'C': 1.0149999999999999})	0,469482745	0,000445087
{KBest({'k': 20})} Support Vector Machine	0,469482745	0,000445087
{KBest({'k': 20})} DummyClassifier({'strategy': 'constant'. 'constant': 0})	0,469482745	0,000445087
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,469269417	0,000460727
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,46916146	0,000973171
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,469018079	0,004117554
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,46798396	0,001687902

– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,467561891	0,002255264
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,464648198	0,003535505
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,461830607	0,041793849
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,454007794	0,04188171
{Fdr} Perceptron({'n_iter': 15})	0,446386682	0,080770364
{Fdr} Perceptron({'n_iter': 20})	0,444971449	0,128120788
{KBest({'k': 100})} Perceptron({'n_iter': 20})	0,442773538	0,121874591
{KBest({'k': 10})} Perceptron({'n_iter': 5})	0,432698875	0,116182168
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,430783347	0,141383063
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,427371825	0,150656061
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,423439196	0,126524624
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,42338327	0,105073537
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,422280589	0,126247249
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,419741023	0,140834112
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,418697606	0,156138493
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,41820626	0,143326154
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,418112351	0,156169049
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,417362454	0,148626079
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,415063865	0,099793436
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,4126201	0,166855886
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,409380168	0,126991713
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,409271461	0,125884762
{KBest({'k': 90})} Perceptron({'n_iter': 20})	0,407979042	0,165998784
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,407661674	0,164005403
{KBest({'k': 90})} Perceptron({'n_iter': 15})	0,405478669	0,150868079
{KBest({'k': 80})} Perceptron({'n_iter': 20})	0,404109747	0,162043732
{Fdr} Perceptron({'n_iter': 5})	0,401781034	0,142522951
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,398174912	0,156420253
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,397378763	0,157854402
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,395915005	0,154885795
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,395915005	0,154885795
{KBest({'k': 10})} Perceptron({'n_iter': 20})	0,395915005	0,154885795
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,395915005	0,154885795
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,395915005	0,154885795
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,395915005	0,154885795
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,395915005	0,154885795
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,395915005	0,154885795
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,395915005	0,154885795
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,395915005	0,154885795
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,395915005	0,154885795
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,395915005	0,154885795
{KBest({'k': 80})} Perceptron({'n_iter': 5})	0,395915005	0,154885795
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,395915005	0,154885795
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,395915005	0,154885795
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,395915005	0,154885795
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,395915005	0,154885795
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,395915005	0,154885795
{KBest({'k': 20})} Perceptron({'n_iter': 5})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,395915005	0,154885795
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,395915005	0,154885795
{KBest({'k': 90})} Perceptron({'n_iter': 5})	0,395808341	0,154829556
{KBest({'k': 100})} Perceptron({'n_iter': 5})	0,39559372	0,154718563
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,395490255	0,154663724
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,395489778	0,154662856
{Variance Threshold({'threshold': 0.10})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{Variance Threshold({'threshold': 0.25})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{Dummy Selector} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{KBest({'k': 100})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{KBest({'k': 10})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{KBest({'k': 90})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{Fdr} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{KBest({'k': 80})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{Variance Threshold({'threshold': 0.15})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{Variance Threshold({'threshold': 0.05})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891
{KBest({'k': 20})} DummyClassifier({'strategy': 'uniform'})	0,395439504	0,018216891

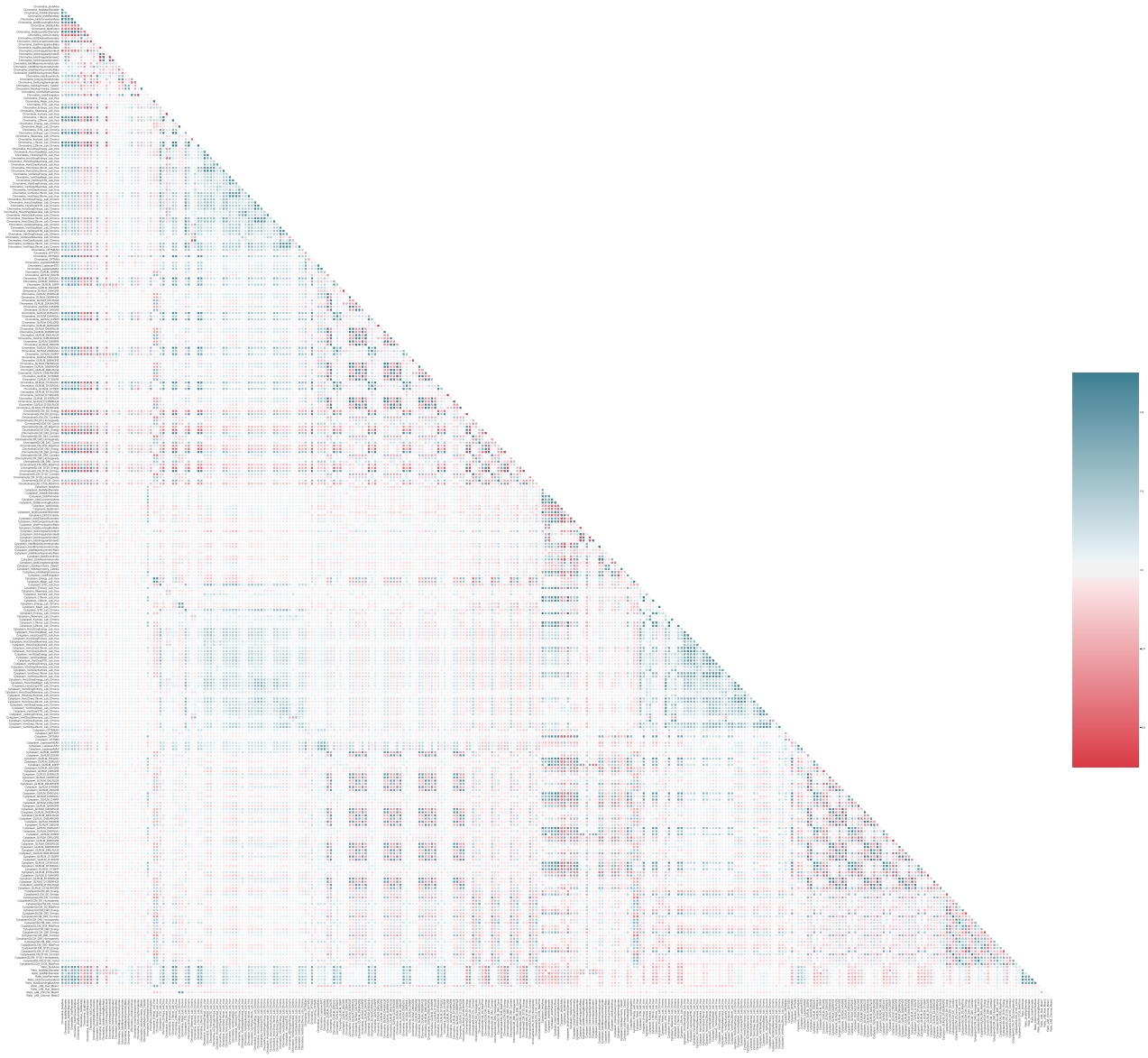
– Continued From Previous Page –

Experiment	Avg F_1 score	σ
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,392921621	0,15331186
{KBest({'k': 20})} Perceptron({'n_iter': 20})	0,384613487	0,152119395
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,38450085	0,142865213
{Fdr} Perceptron({'n_iter': 10})	0,384477665	0,197446247
{KBest({'k': 90})} Perceptron({'n_iter': 10})	0,377698092	0,190586549
{KBest({'k': 100})} Perceptron({'n_iter': 15})	0,370283479	0,173413799
{KBest({'k': 100})} Perceptron({'n_iter': 10})	0,365989115	0,181536352
{KBest({'k': 20})} Perceptron({'n_iter': 10})	0,362949115	0,179153743
{KBest({'k': 80})} Perceptron({'n_iter': 15})	0,334149398	0,192752776
{KBest({'k': 80})} Perceptron({'n_iter': 10})	0,327463855	0,192443662
{KBest({'k': 10})} Perceptron({'n_iter': 10})	0,323608837	0,18868481
{KBest({'k': 20})} Perceptron({'n_iter': 15})	0,322773226	0,179779351
{KBest({'k': 10})} Perceptron({'n_iter': 15})	0,322347265	0,189609816
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,284812191	0,130859645
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,281170581	0,127014598
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,278947538	0,127148639
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,278118964	0,126329495
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,267863618	0,105296381
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,260898867	0,085451008
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,258799857	0,123794438
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,247529796	0,129764569
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,234971423	0,141206009
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,233769367	0,132890214
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,224485725	0,063349178
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,224247299	0,132064713
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,224209406	0,153487894
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,223957969	0,137723029
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,219661085	0,082929351
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,218254375	0,143758166
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,216864249	0,045231643
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,214924287	0,136373647
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,211460956	0,159611845
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,209129685	0,134108172
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,208357348	0,142729531
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,206586821	0,147231651
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,196868019	0,156233835
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,194156252	0,028414956
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,19124345	0,143733353
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,18336005	0,151185275
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,181073651	0,163633572
{Variance Threshold({'threshold': 0.10})} Naive Bayes	0,18074826	0,011561795
{Variance Threshold({'threshold': 0.25})} Naive Bayes	0,18074826	0,011561795
{Dummy Selector} Naive Bayes	0,18074826	0,011561795
{Variance Threshold({'threshold': 0.15})} Naive Bayes	0,18074826	0,011561795
{Variance Threshold({'threshold': 0.05})} Naive Bayes	0,18074826	0,011561795
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,180264709	0,042515684
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,179856611	0,161149574
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,179208477	0,159946114
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,1782637	0,155840018
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,17799034	0,157339225
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,177121989	0,157652612
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,176742111	0,15450106
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,176742111	0,15450106
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,176742111	0,15450106
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 30})	0,176742111	0,15450106
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,176742111	0,15450106
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,176742111	0,15450106
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,176742111	0,15450106
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,176742111	0,15450106
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,176742111	0,15450106
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,176742111	0,15450106
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 50})	0,176742111	0,15450106
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,176742111	0,15450106
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,176742111	0,15450106
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,176742111	0,15450106
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,176742111	0,15450106
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,176742111	0,15450106
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,176742111	0,15450106
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 100})	0,176314161	0,153599721

– Continued From Previous Page –

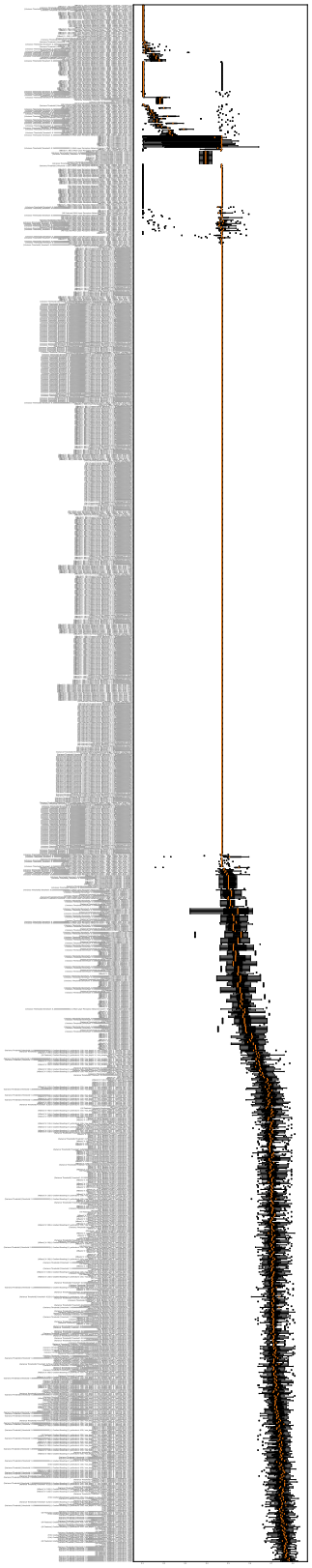
Experiment	Avg F_1 score	σ
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,167624825	0,034118276
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 20})	0,165914006	0,033039078
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,160234819	0,01452588
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,157657101	0,017789123
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,157539697	0,024710104
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,157108693	0,019795299
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 190})	0,150519781	0,01883461
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,148506256	0,087103603
{Variance Threshold({'threshold': 0.10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,142497355	0,022625988
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,133466105	0,044375526
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,132505435	0,010982239
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,124750373	0,016533776
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 140})	0,124490253	0,037298025
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,120869741	0,007343006
{Variance Threshold({'threshold': 0.15})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 70})	0,116285275	0,026580324
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,114053668	0,010762714
{Variance Threshold({'threshold': 0.25})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 110})	0,112309174	0,00512213
{Dummy Selector} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,103633362	0,002225181
{Variance Threshold({'threshold': 0.10})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{Variance Threshold({'threshold': 0.25})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{Dummy Selector} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{KBest({'k': 100})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,10317437	0,001271981
{KBest({'k': 100})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,10317437	0,001271981
{KBest({'k': 10})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,10317437	0,001271981
{KBest({'k': 10})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,10317437	0,001271981
{KBest({'k': 90})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 180})	0,10317437	0,001271981
{KBest({'k': 90})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,10317437	0,001271981
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,10317437	0,001271981
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 90})	0,10317437	0,001271981
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 130})	0,10317437	0,001271981
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 160})	0,10317437	0,001271981
{Fdr} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,10317437	0,001271981
{Fdr} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 40})	0,10317437	0,001271981
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 60})	0,10317437	0,001271981
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 80})	0,10317437	0,001271981
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 120})	0,10317437	0,001271981
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 150})	0,10317437	0,001271981
{KBest({'k': 80})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 170})	0,10317437	0,001271981
{KBest({'k': 80})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{Variance Threshold({'threshold': 0.15})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{Variance Threshold({'threshold': 0.05})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 10})	0,10317437	0,001271981
{Variance Threshold({'threshold': 0.05})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981
{KBest({'k': 20})} Multi-Layer Perceptron Network({'solver': 'lbfgs'. 'hidden_layer_sizes': 200})	0,10317437	0,001271981
{KBest({'k': 20})} DummyClassifier({'strategy': 'constant'. 'constant': 1})	0,10317437	0,001271981

B.2 Features Correlation Matrix



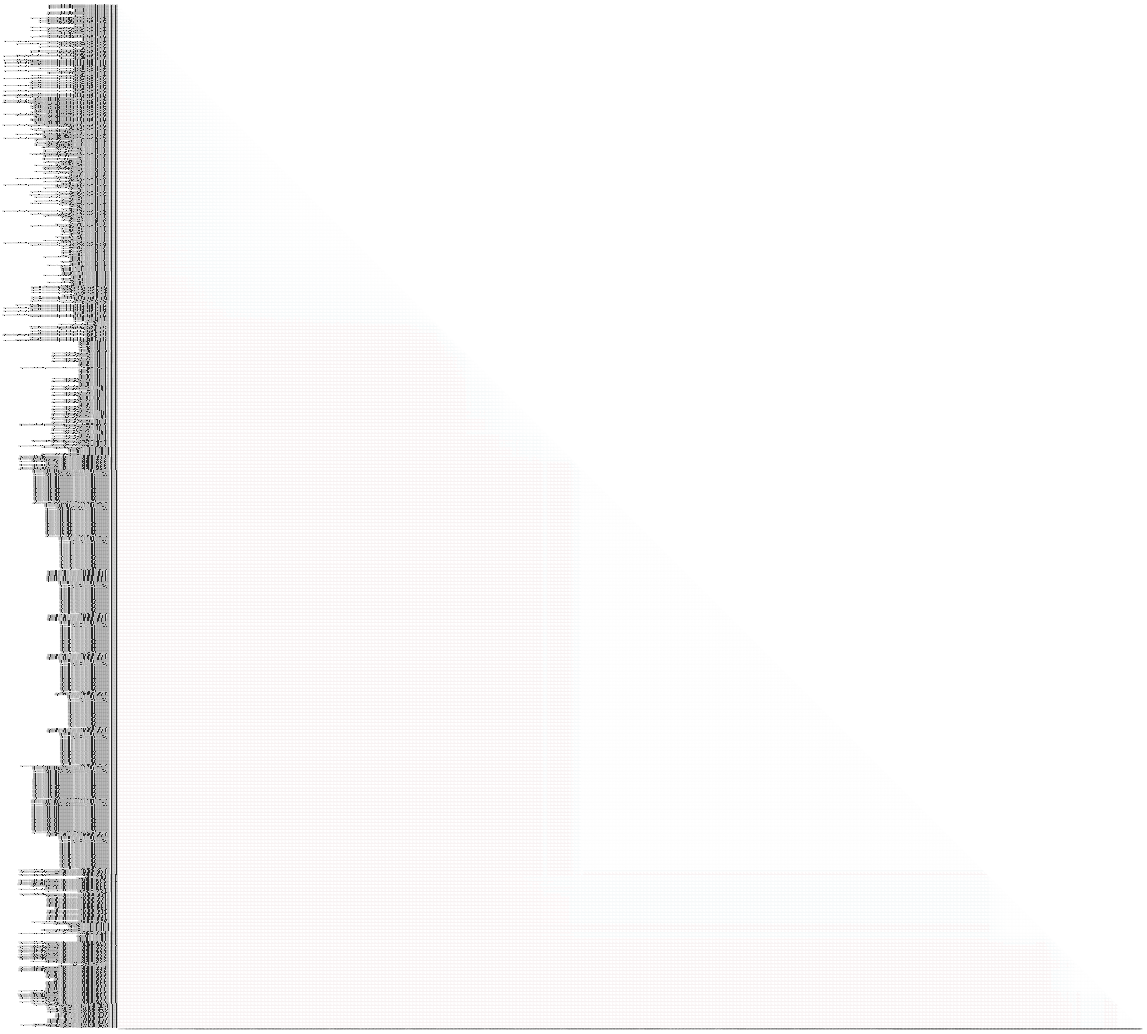
A full resolution of this correlation matrix can be found at: http://tinyurl.com/ybgysv44.

B.3 Data Models Box Plot



The full resolution of box plot graph can be found at: http://tinyurl.com/ycscfr4t.

B.4 Data Models *TTest* Heatmap Matrix



The full heatmap matrix and box plot graph can be found at: http://tinyurl.com/ya6ff7rf.